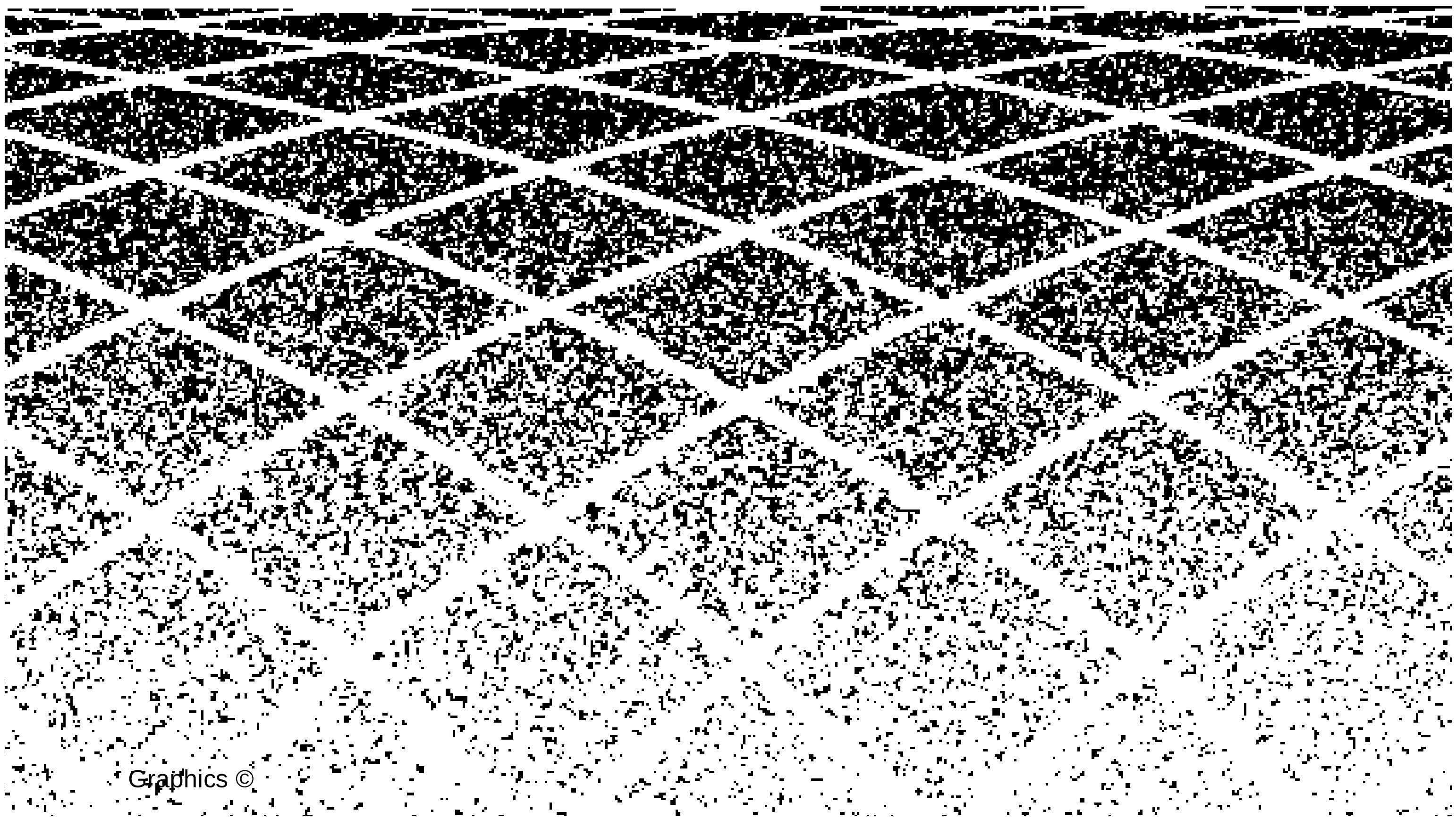
585-350-209
Issue 1
October, 1993

AT&T

CONVERSANT Voice Information System (VIS) Quick Reference of Commands



Blank Page

Contents

Contents

Contents

Contents

Contents

Contents

Contents

Contents

Contents

Contents

Contents

Contents

Contents

Contents

Contents

About This Book

Purpose

This book is designed as a quick reference of commands used with the CONVERSANT Voice Information System (VIS).

How to Use This Book

This book provides an alphabetical listing of the CONVERSANT VIS commands. For each command listed, the following information is provided (where applicable):

- Name — This section provides the name and purpose of the command.
- Synopsis — This section summarizes the usage of the program being described.
- Description — This section discusses how to use the command.
- Example(s) — This section provides examples of the usage of the command, where appropriate.
- Notes — This section provides information that may be helpful under the particular circumstances described.
- Warning — This section discusses the limits or boundaries of the respective command, where appropriate.
- Caveats — This section points out possible pitfalls to consider when using the command.
- Return Values — This section describes the values returned if the command is successful or if the command failed.
- Files —This section gives the file names that are built into the program or the command.
- See Also — This section offers pointers to related information or related commands.

This book also provides a list of abbreviations used in CONVERSANT VIS documentation and a cross-referenced index listing the principal subjects contained in this book.

Conventions Used in This Book

The following typographic conventions are used in this book:

- The word “enter” means to type a value and press ENTER. For example, an instruction to type y and press ENTER is shown as

  Enter **y** to continue.

- The word “select” is used to mean the following: move to the desired menu item using the arrow keys and press ENTER.

- Terminal keys are shown in rounded boxes. For example, an instruction to press the enter key is shown as

  Press ENTER.

- Function keys (also known as “soft” keys) are shown in boxes followed by the actual name of the key on the keyboard in parentheses. For example, an instruction to press the choices key is shown as

  Press CHOICES (F3).

- Two or three keys that you press at the same time (that is, you hold down the first key while pressing the second and/or third key) are shown as a series of rounded boxes. For example, an instruction to press and hold ALT while typing the letter d is shown as

  Press ALT d.

- Information that is displayed on your terminal screen – including screen displays, field names, prompts, and error messages – is shown in `type-writer-style constant-width` type; for example

  ```
  Installation is in progress -- do not remove the
  floppy disk.
  ```

- Information that you enter from your terminal keyboard is shown in bold type, for example

  Enter **root** at the `Console Login` prompt.

- Command and file names and their parameters are shown in **bold** type. Variable parameters are shown in ***bold italic*** type when they are part of a user input and in *regular italic* type when they are not. All are illustrated in the following example:

  Use the **print** command to print your report. The command syntax is **print *reportname***, where reportname is the name of the report to be printed.

- Square brackets [] around an argument indicate that the argument is optional.
- List of options for a single argument are divided by vertical bars (a|b|c).
- Ellipses ... are used to show that the previous argument may be repeated.

Related Resources

The following books contain information related to the commands that appear in this book:

- *CONVERSANT VIS Version 4.0 Maintenance*, 585-350-112
- *CONVERSANT VIS Version 4.0 Application Development*, 585-350-208
- *CONVERSANT VIS Version 4.0 Operations*, 585-350-703

This book may be used in conjunction with all other standard and feature package in the CONVERSANT VIS library. Refer to *CONVERSANT VIS Documentation Guide*, 585-350-002, for a complete list of VIS documentation.

Trademarks and Service Marks

The following trademarked products are mentioned in this book:

- CONVERSANT is a registered trademark of AT&T.
- AUDIX and Voice Power are trademarks of AT&T.
- CLEO and DataTalker are trademarks of CLEO Communications.
- IBM is a registered trademark of International Business Machines.
- UNIX is a registered trademark of UNIX System Laboratories, Inc.
- ORACLE, SQL*Net, and SQL*Plus are registered trademarks of the Oracle Corporation.

How to Make Comments About This Book

A reader comment card is behind the title page of this book. While we have tried to make this book fit your needs, we are interested in your suggestions for improving it and urge you to complete and return a reader comment card.

If the reader comment card has been removed from this book, please send your comments to:

AT&T
Product Documentation Development
Room 22-2C11
11900 North Pecos Street
Denver, Colorado 80234

Please include the name and order number of this document.

Summary of Commands

1

This chapter contain summaries of the commands in the CONVERSANT Voice Information System (VIS) Version 4.0 documentation. The following table provides an alphabetical listing and brief description for all the commands in this book. Use the tables to locate the command needed to perform a particular task. Then, find the more annotated command page provided later in the chapter.

Table 1-1. Command Synopsis

Command	Function
"3270dip_off"	Turns off the 3270 DIP
"3270dip_on"	Turns on the 3270 DIP
"add"	Adds a phrase to the speech database
"add_device"	Adds a new device to the **/vs/data/device_data** file
"addboard"	Adds or modifies the hardware configuration informa-tion for a 3270 card to the system.
"addmsg"	Allows for the addition of explanation texts for error messages prior to Version 3.1.
"addspdisk"	Partitions a second hard disk for speech
"annotate"	Annotates a TSM trace stream with a message
"assign card/ channel"	Assigns a group number to a card or channel
"assign ser-vice"	Assigns an installed application to a DNIS number or to a channel
"attach"	Attaches a unit
"audit"	Verifies the format of the speech file systems
"autoreboot"	Changes or displays the parameters associated with the **"autoreboot"** feature
"backup_appl"	Backs up an application if enhanced file transfer is installed
"bbs"	Reports the status of the voice system Bulletin Board
"buildfs"	Builds a speech file system on a specified disk slice
"ccarpt"	Generates a call classification data summary report
"cddrpt"	Generates a call data detail report
"cdsrpt"	Generates a call data summary report for a specific date
"change_devic e"	Changes the name of a device in the **/vs/data/ device_data** file

Table 1-1. Command Synopsis

Command	Function
"checktf"	Checks for the existence of talkfiles in the voice system
"cmgrtool"	Provides trace utilities for the 3270 Protocol Communications Manager (CMGR)
"configure"	Determines the allocation of resources for all devices to be included in the system configuration for a given hardware platform
"copy"	Copies a phrase from the speech database to another UNIX file
"cpuType"	Returns the type of CPU used in the system
"cvis_mainmenu"	Accesses the topmost CONVERSANT administrative menu
"cvis_menu"	Accesses the CONVERSANT Voice System Administration
"dbcheck"	Checks the resources available in the database
"dbfrag"	Lists fragmentation information on the database
"dbfree"	Checks the space available in the database by partition
"dbused"	Provides database use by **oracle** user
"delete card/ channel"	Removes a card or channel from a service or an equipment group
"delete eqpgrp"	Removes an equipment group
"delete service"	Removes the telephone number for a script from a group
"detach"	Places a unit in the nonexistent state
"diagnose bus"	Tests a bus while it is in service
"diagnose card"	Tests a card while it is in service
"dip_int"	Sends a DIP interrupt to a script on a channel or a range or channels

Table 1-1. Command Synopsis

Command	Function
"display card"	Displays information about specified cards
"display chan-nel"	Displays channel information
"display eqpgrp"	Displays an equipment group report
"display mes-sages"	Displays system messages
"display ser-vice"	Lists all valid services or scripts
"displaypkg"	Lists the software packages installed on the VIS
"dscope"	Displays the trace data collected by **"cmgrtool"**
"edExplain"	Edits the explanation text for one or more message tags
"erase"	Deletes a phrase from the speech database
"etStub"	Reads the IPC message queue for error messages that use the ET process
"explain"	Displays on-line error message explanations
"fixLogFile"	Upgrades existing logging files
"get_config"	Retrieves the **/vs/data/conf_data** file from a floppy disk
"gse_add"	Transfers a speech phrase from a UNIX file to the speech database in the Graphical Speech Editor (GSE) format
"gse_addpl"	Adds (restores) phrases to a specific speech pool from UNIX files in the GSE format
"gse_copy"	Extracts a speech phrase from the VIS speech file system to a UNIX file in the GSE format
"gse_copypl"	Copies multiple speech phrases from the speech file system in the GSE format
"hassign"	Assigns host services to host sessions
"hdelete"	Removes host services from host sessions

Table 1-1. Command Synopsis

Command	Function
"hdisplay"	Shows host applications that have been successfully verified and installed
"headFIX"	Converts ET message rules header files to the logger/ alerter environment
"hfree"	Releases host sessions from Script Builder host application assignments
"hlogin"	Runs the login sequence of a host script
"hlogout"	Runs the log out sequence of the host script
"hnewscript"	Installs a changed host script
"host_cfg"	Translates an ASCII configuration file into a properly formatted binary configuration file
"hsend"	Sends a file to the host via CONVERSANT file transfer
"hspy"	Displays a screen currently present on the specified host session
"hstatus"	Shows the current status of the host sessions
"iCk, iCkAdmin"	Performs various integrity checks based on the rules in a script file
"install_appl"	Installs an application if enhanced file transfer is installed
"install_remote"	Sets up and activates the remote login port and modem
"install_sw"	Installs a software package if enhanced file transfer is installed
"installpkg"	Installs a software package
"into_et"	Sends error messages to ET
"lComp"	Combines message files to produce compressed and expanded format files
"list"	Lists the directory entries for specific phrases
"load_bin"	Initializes the 3270 host card
"logCat"	Reads compressed logging files and outputs human readable messages

Table 1-1. Command Synopsis

Command	Function
"logdaemon, reinitLog"	Sends received messages to the appropriate destinations
"logDstPri"	Creates the shared memory containing the dynamic destinations and priorities of logging messages using **logMsg**
"logFmt"	Displays and changes the parameters used to display messages and explanation texts
"logit"	Logs arbitrary messages to the logging files
"logTest"	Generates an arbitrary list of logging messages to be sent to logdaemon
"lsboard"	Displays configuration information for all 3270 cards on the system
"mkAlerter"	Reads an alerter description and generates the code that implements the description
"mkerr"	Converts ET message rules to the logger/alerter environment
"mkETrules"	Produces a valid **etStub.rules** file from one or more errors files used to control the ET process
"mkheader"	Allocates user memory for script variables
"mkimage"	Performs a system backup
"mkMsg"	Converts ET message rules to files for use in the logger/alerter environment
"msgadm"	Facilitates the administration of system messages
"newscript"	Updates the changes to all currently assigned scripts
"remove"	Places a unit in the manual-out-of-service (MANOOS) state
"remove_appl"	Removes an application if enhanced file transfer is installed
"remove_device"	Removes a device from the **/vs/data/device_data** file
"remove_remote"	Removes the remote login modem from the auto-answer mode

Table 1-1. Command Synopsis

Command	Function
"remove_sw"	Removes an installed package if enhanced file transfer is installed
"removepkg"	Removes a software package
"restore"	Restores a unit to the in-service (INSERV) state
"restore_appl"	Restores an application if enhanced file transfer is installed
"rmboard"	Removes the hardware configuration information for a 3270 card from the system
"save_config"	Saves the **/vs/data/conf_data** to floppy disk
"sb_te"	Invokes the 3270 terminal emulator
"sb_trace"	Displays trace message and screens being sent between Script Buillder applications and the host mainframe for the specified host channel
"sccsDaemon"	Distributes messages to the SCCS or CompuLert systems and to the alarm relay unit (ARU); Can be used to send commands from the user to the daemon process
"show_config"	Displays and prints to file the valid or incomplete VIS configuration
"show_devices "	Displays and prints to file all devices and their attributes as represented in the **/vs/data/device_data**
"soft_disc"	Sends a disconnect to a script on a channel or channels.
"soft_szr"	Starts a script on a channel
"spch2unix"	Converts speech slices into UNIX file systems
"spCtlFlags"	Sets and clears flags used to control the behavior on SP executive pack files as they run on an SP card
"spres"	Restores speech from a backup
"spsav"	Saves speech to backup
"spStatus"	Displays information about the pack file running on an SP card

Table 1-1. Command Synopsis

Command	Function
"spVrsion"	Prints the version of the SP driver currently installed on a machine
"start_vs"	Brings the voice system up to a fully operational state
"stop_vs"	Stops the voice system software gracefully
"sysmon"	Executes a program that monitors incoming telephone lines and the associated cards to see that they are functional
"tas"	Executes the transaction assembler program to assemble script instructions
"trace"	Outputs trace messages for the specified processes and channels
"trarpt"	Generates the call traffic report file systems
"upg"	Provides automated assistance in upgrading a CON-VERSANT VIS machine to Version 4.0
"vdf"	Lists the number of free 16-kbyte blocks on each speech disk slice
"vsdisable"	Disables the automatic restarting of the voice system
"vsenable"	Enables the automatic starting of the voice system at system reboot
"vusage"	Displays the current load on the voice system
"xferdip_off"	Deactivates the bridging capability
"xferdip_on"	Activates the bridging capability

3270dip_off

Name

This command turns off the 3270 data interface process (DIP) if it is not already.

Synopsis

3270dip_off

Description

The **3270dip_off** command deactivates the 3270 DIP the next time the voice system is started.

NOTE:
You must stop and start the voice system for this command to take effect.

Files

/vs/data/HOST3270
/etc/inittab

Example

The following example turns off the 3270 DIP.

```
3270dip_off
```

See Also

"3270dip_on"

3270dip_on

Name

This command turns on the 3270 data interface process (DIP) if it is not already.

Synopsis

3270dip_on

Description

The **3270dip_on** command activates the 3270 DIP the next time the voice system is started.

NOTE:
You must stop and start the voice system for this command to take effect.

Example

The following example turns on the 3270 DIP.

```
3270dip_on
```

See Also

"3270dip_off"

add

Name

This command adds a phase to the speech databases.

Synopsis

add phrase <*phrase numbe*r> to talkfile <*talkfile number*> from <*file_name*>

Description

This command adds phrases to the specified talkfile that were previously extracted from another talkfile using the **"copy"** command. The path name for the file may be the full path name or the relative path name. If no path is specified, the file is created in the current working directory. If you are not in the directory in which the phrase to be added is stored, be sure to give the full path name for the talkfile and the source file.

If the phrases already exists, the system displays the following message:

```
Phrase <phrase_number> already exists in talk file
<talk file number> Do you want to overwrite existing
phrase? (y/n)?
```

If an error occurs, system messages are printed on the controller screen. The source file may be a full path name or a relative path name. Refer to Chapter 3, “System Message Listings,” of *CONVERSANT VIS Version 4.0 Maintenance*, 585-350-112, for information about how to respond to a system message.

Files

/speech/talk/*.pl

Example

The following example adds phrase number 275 to talkfile 25 from the directory **/tmp/junk**.

```
add phrase 275 to talkfile 25 from /tmp/junk
```

The following example adds phrase 104 to talkfile 18 from the directory **/speech/talk**.

```
add phrase 104 to talkfile 18 from /speech/talk
```

See Also

"copy"
"erase"
"list"

add_device

Name

This command adds a new device to the **/vs/data/device_data** file.

Synopsis

/vs/bin/util/add_device

Description

The **add_device** command is an interactive command which allows the user to add a new device to the **/vs/data/device_data** file. Once a new device is added to this file, it appears in the device menus of the **/vs/bin/util/configure** program and may be selected for possible configuration in a VIS machine.

The **add_device** program prompts for all attributes that must be specified to make a new device entry. If there are no special attributes or rules associated with the new device, executing **add_device** is all that is necessary to add complete support of the device to the **"configure"** program.

The **add_device** program prompts for the following attributes:

NOTE:
A “-” (minus) in front of the attribute’s description indicates a required attributed, whereas an “*” (asterisk) indicates an attribute to be entered only if applicable. If an attribute is not applicable, entering **q** (for quit) when the prompt for that attribute is displayed skips that attribute.

- - Device Description: An 80-character limit. This attribute should list generic name, comcodes, list numbers, part number, etc.
- - User Mnemonic Name: A 20-character limit. This attribute should be the name by which all presentations of this device appear in the **"configure"** program’s menus. This name must be unique from all other user mnemonic names.
- - Program Mnemonic Name: An 8-character limit. This attribute should be the name by which the **"configure"** program will internally refer to this device. This name must be unique from all other program mnemonic names.

- - Device Type: An 8-character limit. This can usually be the same as the program mnemonic name. However, in some cases, cards with different program mnemonic names are serviced by the same software driver. In this case they may share a hardware resource such as an interrupt.

  An example is the Tip/Ring (T/R) interface cards IVP6, IVP4, and VRS6. All three can share the same interrupt and are serviced by the same driver. By specifying type as TR for all three of these cards, the **"configure"** program is informed of this fact.

- * Interrupts: Each interrupt that is usable by the device should be entered one at a time as prompted for. The interrupts should be entered in priority order, that is, the preferred interrupt for this device should be entered first, second choice next, etc. After all interrupts are entered, the user is prompted to indicate whether all devices of this device's type (as described above) can share the same interrupt. Response is **y** or **n**.

- * Base IO address: Each base input/output (IO) address that is usable by the device should be entered one at a time as prompted for. The addresses should be entered in priority order, that is, the preferred address for this device should be entered first, second choice next, etc. Input must be specified in hex.

  After all addresses are entered, the user is prompted to enter the number of contiguous bytes of IO space that is required from the base address for this device. The number is expected in decimal. Note that the device can have only one range of IO addresses starting at the base. If the device does not meet this criteria, the **"configure"** program is not able to accurately determine configurations which include this device.

  After the above information is entered, the user is prompted to indicate whether all devices of this device's type (as described above) can share the same IO address. Response is **y** or **n**.

- * Base RAM address: Each base random access memory (RAM) address that is usable by the device should be entered one at a time as prompted for. The addresses should be entered in priority order, that is, the preferred address for this device should be entered first, second choice next, etc. Input must be in hex.

  After all addresses are entered, the user is prompted to enter the number of contiguous kbytes of RAM space that is required from the base address for this device. The number is expected in decimal and is to be expressed in kbytes. Note that the device can have only one range of RAM addresses starting at the base. If the device does not meet this criteria, the **"configure"** program will not be able to accurately determine configurations which include this device.

  After the above information is entered, the user is prompted to indicate whether all devices of this device's type (as described above) can share the same RAM address. Response is **y** or **n**.

The user is also prompted to enter the data transfer bit rate supported by the device. This is usually (but not necessarily, check device documentation to be sure) the same as the slot type (8 or 16 bit).

This attribute is required because devices which can support only 8-bit data transfers may not reside in the same 128-kbyte block of RAM as devices which support 16 bit data transfers. If the new device supports transfers data transfers 8 bits at a time, enter 8. If it supports transfers 16 bits at a time, enter 16. If the **"configure"** program is not to check this attribute, enter **x** for do not care (this should be avoided).

- * DMA Channel: Each direct memory address (DMA) channel that is usable by the device should be entered one at a time as prompted for. The DMA channels should be entered in priority order, that is, the preferred channel for this device should be entered first, second choice next, etc.
- * Slot Type: The slot type required for the new device must be entered. 8, 16, or 32 bit slots are supported. Enter **q** if the device does not require a slot. Confirmation is required if **q** is entered.
- * Miscellaneous Attributes: Answer **y** or **n** to the prompts for miscellaneous attributes:

  a. CONVERSANT VIS System Board: Enter **y** if the card is a CONVERSANT network interface card or speech processing card. Enter **n** otherwise.

  b. Serial Port Required Flag: Enter **y** if the device requires a serial port. Enter **n** otherwise.

  c. Parallel Port Required Flag: Enter **y** if the device requires a parallel port. Enter **n** otherwise.

Upon entering all requested information, the user has the option of confirming it or changing some part of it. If confirmed, the information is entered into the **/vs/data/device_data** file. The new device appears in the **"configure"** program's menu the next time it is invoked.

Files

/vs/data/device_data

See Also

"configure"
"change_device"
"remove_device"
"show_devices"
"show_config"
"save_config"
"get_config"

Notes

In order for the **"configure"** program to determine accurate configurations, it is extremely important for all pertinent information to be accurately entered into the **device_data** file concerning a new device. When adding a new device, consult the hardware documentation concerning the device and be certain that the information prompted for by the **"add_device"** program is entered completely and accurately.

addboard

Name

This command adds or modifies the hardware configuration information for a 3270 card to the system.

Synopsis

addboard

Description

The **addboard** command enables you to add or modify hardware configuration information for a single 3270 host card to the system. You may physically add the 3270 hardware before or after using the **addboard** command. If you are reconfiguring an existing 3270 card, you should use **"rmboard"** and then **addboard**. This command invokes a program prompting you for all required information.

Installing a new 3270 host card requires that VIS be shut down and then rebooted. Consequently, after you perform the **addboard** command, you must execute **"stop_vs"** and **"start_vs"** commands from the UNIX system command line to stop and restart the voice system and automatically activate the new hardware information for the selected 3270 host card.

You must be logged in as **root** (superuser) before using the **addboard** command.

If the **addboard** complains of an I/O address conflict, the problem could be an I/O address problem or a RAM address problem. Check the I/O and RAM address on the host card or cards using the **"lsboard"** command to be sure that no other device is using those addresses.

See Also

"lsboard"
"rmboard"

addmsg

Name

This command is the compatibility program which allows for the addition of explanation texts for error messages prior to Version 3.1.

Synopsis

addmsg

Description

The **addmsg** command is the program that added and/or modified explanations for error numbers in VIS systems prior to Version 3.1 (V3.1). A compatibility version is provided for VIS 3.1. Use **"edExplain"** in Version 3.1.

addmsg takes one or more repetitions of the form:

> {**error-number}**
> **one or more lines of explanation text**

The *{error-number}* must be strictly digits. This identifies error numbers for pre-Version 3.1. The explanation follows immediately on one or more lines and is terminated by a line containing only a period.

Input is terminated by entering CTRL D when prompted for the next error number.

As each explanation is entered, it is placed in an individual file in the appropriate /**gendb/data/explain/*{dir}*** directory, where *{dir}* is the first digit of the error number.

Files

/gendb/data/explain/[0-9]/{err-num}	# Explanation file
/tmp/add[mM]sg.{PID}	# Two temporary files

Diagnostics

addmsg returns the number of faulty explanations entered. Any attempt to enter an explanation with no lines in it is considered to be an error. The value 0 is returned when **addmsg** is completely successful.

Examples

The following example is input for message text explanations from a tty:

```
# addmsg
What’s the error code number? (Press ‘<Ctrl>-d’ to
quit) 5620
Type in message. (End by a . on line by itself)
5620 indicates that a bad record has been received.
.
What’s the error code number? (Press ‘<Ctrl>-d’ to
quit)  <CTRL-D>
```

The following example displays input information from a file. If input comes from a file, a “here” file, or via a pipe from another process, no prompts are issued:

```
# addmsg <<!
5620
5620 indicates that a bad record has been received.
.
5621
5621 indicates that there is no more room for records.
.
!
```

See Also

"edExplain"

addspdisk

Name

This command automatically partitions a second hard disk for speech.

Synopsis

addspdisk

Description

The **addspdisk** command automatically partitions a second hard disk to hold speech.

The **addspdisk** partitions the hard disk into one UNIX partition, then performs a bad track analysis on the partition. It then partitions the new UNIX partition according to the following:

- If a speech file system exists on disk 0, two speech file system partitions are created on disk 1. The first is the same size as the speech partition on disk 0 and the second contains the rest of the disk.
- If a speech file system does not exist on disk 0, one speech file system partition containing the entire disk is created on disk 1.

The command then updates the **/etc/partitions** and **/vs/data/fslist** files to reflect the partitioning.

Note that before the second disk can be added, you may have to use the low-level formatting procedures provided in Chapter 10, “Adding Additional Hardware,” of the hardware book specific to your platform.

Example

The following example partitions hard disk 1 for speech.

```
addspdisk
```

Files

/etc/partitions
/vs/data/fslist

annotate

Name

This command annotates the TSM trace stream with a message.

Synopsis

annotate ***[channel]*** ***<"message">***

Description

The **annotate** command sends a message to TSM requesting that the given message be put into TSM's trace stream. This command is useful for testing and debugging scripts. It resides in **/vs/bin/tools**.

If a channel is specified, the message is associated with the channel's trace stream. The message must be fewer than 160 characters.

The **annotate** trace message is displayed in the trace output if a trace is running when the **annotate** command is executed. If no **"trace"** command is running, the annotate trace message is discarded.

Example

The following example sends a message to TSM to put the message "This is test 1 for channel 1" in channel one's trace stream.

```
annotate 1 "This is test 1 for channel 1"
```

assign card/channel

Name

The **assign card** command assigns a group number to a card.
The **assign channel** command assigns group number to a channel.

Synopsis

assign card ***<number>*** **to** ***[eqpgrp] <group number> [grpname]***

assign channel ***<number>*** **to** ***[eqpgrp] <group number> [grpname]***

Description

The **assign** command is used when a system is installed, the number of channels or cards changes, scripts are added or deleted, telephone numbers change, or the user wants to reconfigure the system. The system uses the card and channel assignments to route an incoming call to the group.

The parameters that can be used with the **assign** command are:

- *number* — The channel number (a single card or channel number, a range of card or channel numbers specified m–n, or the word “all’ for all card or channel numbers)
- *group_type* — The “eqpgrp” when assigning to an equipment group
- *group number* — The number of the equipment group or service group
- *name number* — An optional character string that can be associated with “grp’“

Reference to a nonexistent channel or nonexistent group in this command causes it to fail.

Examples

The following example assigns channels 0 through 47 to equipment group 1.

```
assign chan 0-47 to eqpgrp 1
```

assign service

Name

This command assigns an installed application to either a DNIS number or directly to a channel.

Synopsis

assign service <*application_name*> to <*chan|dnis*> <c*hanlist|phonelist*>

Description

The **assign service** command is used to assign voice script names to either a set of channels or to a DNIS number. Service should be assigned after an application has been verified and installed, the number of channels changes or the system is reconfigured. Use the **"display service"** command to see a list of valid service names.

The *phone number* allows a specific incoming T1 channel to be associated with a specific voice script. If an incoming phone number is not specifically associated with a voice script, the call is handled by the voice script whose phone number is designated as “any.” If the phone number does not match any of the phone numbers and there is no voice script with “any” as the phone number, the call is not answered.

The <*application_name*> is the service name.

Example

This example assigns service stdin (standard in as an arbitrary name for a script) to channel 0.

```
assign service stdin to chan 0
```

This example assigns service stdout (standard out as an arbitrary name for a script) to channel 1.

```
assign service stdout to chan 1
```

This example assigns service dnis to all channels.

```
assign service *DNIS_SVC to chan all
```

See Also

"delete service"
"display service"

attach

Name

This command attaches a unit.

Synopsis

attach *<unit> <number> [-i] [-n]*

Description

The **attach** command is used to attach a card that has been “detached.” The unit (card) is logically attached by changing its permanent state from nonexistent (NONEX) to manual-out-of-service (MANOOS). To put the unit into service, use the **"restore"** command.

The parameters for the **attach** command are:

- *<unit>* — This option identifies the unit; the choices are “channel” or “card.”
- *<number>* — This option specifies the channel or card number, a range of channel or card numbers in the form m–n, or the word “all” for all channel or card numbers. Card numbers are in the form ***card#[.port#]*** where *port#* is a port of the card *card#*. If *port#* is not given, all ports of the card specified are attached. If no card number or channel number is given, a syntax message is displayed.
- *-n* — This option disables prompting from the system whether to wait until a conflict has been resolved (see the *-i* option below) or to terminate the request to **attach**.
- *-i* — This option is used to enable secondary command registration. If T1 diagnostics are being run, this option allows the “attaching” of another card. If *-i* is used and another maintenance command is being run (**"remove", "detach", "attach", "restore"**, or **diagnose**), the request to **attach** is blocked and a message is printed to the screen. If *-i* is not used and any maintenance command is being run, the request to **attach** is blocked and a message is printed to the screen.

  If the command is permitted to run, it is determined if the command is in conflict with another command. A command is in conflict if the card or card associated with it:

  1. Is the T1 card being diagnosed
  2. Will cause a change in the existing TDM bus master assignment

3. Has an interdependency with the T1 card being diagnosed (for example, PRI)

If one of the above conflicts exist and *-n* is not used, the user is asked whether to wait until the conflict is resolved or to terminate the request. If T1 diagnostics are executing on-line tests and a conflict is detected, the **attach** command is blocked. If T1 diagnostics are executing off-line tests and a conflict is detected, the user is asked whether to wait until the conflict is resolved or to terminate the request to **attach**.

To delete out of the command, press DEL. If this does not terminate the command, you may need to press CONTROL ALT DEL simultaneously. If, while running **attach**, you abort the command, a message similar to the following may appear:

```
At the user’s request, administration of the following
cmd(s) has been interrupted.

CARD NUMBERS: <card numbers>

To assure proper operation of the identified card(s),
run diagnostics at the earliest opportunity.
```

It is recommended when **attach** is aborted, diagnostics be run on all cards being administered to ensure they are returned to a fully functional state.

Examples

The following example attaches a card to channel 2.

attach card 2

The following example attaches channels 0 through 2 and channel 5.

attach channel 0-2,5

The following example attaches a card to channel 2, port 1.

attach card 2.1

See Also

"detach"
"restore"
"remove"

audit

Name

This command verifies the format of the speech file systems.

Synopsis

audit ***[-0|-1|-2] [-v|-f] [fsnames]***

Description

NOTE:
Stop the VIS before executing this command.

The **audit** command verifies the format of the speech file systems. Speech file systems can be verified one at a time or all at once. Use this command if errors (bad super block, bad phrase descriptor, or bad phrase) are reported in the speech file system.

The valid entries for the first argument are:

-0	This argument checks the super block only.
-1	This argument checks the super block and phrase descriptors.
-2	This argument checks the super block, phrase descriptors, and phrases

The valid entries for the second argument are:

-v	This argument executes the command in verbose mode.
-f	This argument fixes any errors found.

The *fsnames* argument is a list of file systems to verify. If you execute **audit** without any arguments, it checks the super block, phrase descriptors, and phrases in all speech file systems in the **/vs/data/fslist** file.

Example

In the following example, the VIS verifies the super block and phrase descriptors in the file systems on slice 0s4 and fixes any errors that are found.

```
audit -1 -f /dev/rdsk/0s4
```

autoreboot

Name

This command provides a means of changing or displaying the parameters associated with the VIS automatic reboot feature.

Synopsis

autoreboot ***[enable|disable] [reboots <numbers>] [window <minutes>] [uptime <minutes>]***

autoreboot ***[status|s]***

autoreboot ***[help|h]***

Description

The **autoreboot** command may be used to change parameters associated with the VIS autoreboot feature and to monitor the status of these parameters. The following options are recognized:

- *enable|disable* — This option specifies whether to enable or disable the autoreboot feature. The default is enable.
- *reboots <number>* — This option specifies the number of unanticipated reboots tolerated within the time period specified by *window*. The default is 5.
- *window <minutes>* — This option specifies the time period for the *reboots* parameter. The default is 60 minutes.
- *uptime <minutes>* — This option specifies the amount of time that the system must be in service before the automatic reboot feature is activated. The default is 5 minutes.
- *status* — This option shows the current values of the automatic reboot parameters, plus the number of unanticipated reboots that occurred in the *window* minutes preceding the most recent system boot.

When the automatic reboot feature is enabled and activated, the system automatically reboots after a UNIX panic. The automatic reboot feature is activated as follows:

If there were fewer than *reboots* unanticipated reboots during the *window* minutes prior to the most recent system boot, the automatic reboot feature is activated (if enabled) *uptime* minutes after the most recent system boot.

For example, assume the automatic reboot parameters are set to their default values. A system crash occurs. The system reboots at 8:00. If there were fewer than 5 unanticipated reboots between 7:00 and 8:00, the automatic reboot feature is activated as 8:05. Otherwise, it is activated at 9:00.

An unanticipated reboot is a system boot that occurs after a system crash. A system crash can be caused (for example) by a UNIX panic, a system restart via RESET or a sudden power loss.

Example

The following example enables autoreboot feature and changes *window* to two hours:

```
autoreboot enable window 120
```

Caveat

This command must be run from ksh (KORN shell).

backup_appl

Name

This command backs up an application.

NOTE:
This command is valid only if the Enhanced File Transfer package is installed.

Synopsis

backup_appl ***-n <application name> [-d <database file>] [-t <transaction file>] [-s <speech file>] [-p <path>]***

Description

The **"backup_appl"** command is used to backup a Script Builder application to files on the local machine. The files in each component (database, speech, transaction) of the application are bundled into one cpio file per component. If the cpio file names and path are not specified, default names and a default path are used and all three components are backed up. The following are the default file names for each component:

database	Dbase
speech	Spch
transaction	Trans

The default path to backup all components of a specific application is **/tmp/sb/BkUpAppl/<a*pplication name*>**.

Return Values

If the **backup_appl** command is successful, a 0 value is returned.If any value other than 0 is returned, the **backup_appl** command failed.The following are the possible reasons for failure for the **backup_appl** command:

- The hard disk is low in space.
- You are not logged in as **root** or a super user.
- The command syntax is incorrect.
- The backup tables has failed.
- The backup speech has failed.
- The backup transaction has failed.

Example

The following example backs up the “bank_balance” application using the default names for the transaction, database, and speech.

```
backup_appl -n bank_balance
```

See Also

"install_appl"
"remove_appl"
"restore_appl"

bbs

Name

This command reports status of the voice system Bulletin Board (BB).

Synopsis

bbs ***[-d] [-h] [-l]***

Description

The **bbs** command displays the field values of the BB slots. This information is stored in standard out (stdout). Without any options, information is extracted only from the dynamic portion of the BB and printed in short format. Otherwise the information displayed is controlled by following options:

d	This option prints information about the dynamic portion of the BB (the default).
h	This option prints information about the hardcoded portion of the BB.
l	This option generates a long listing. All fields are displayed.

The column headings and meaning of the columns in the **bbs** listing are given in Table 1-2. In the table, the letter **l** indicates the **long** option, which causes the corresponding heading to appear. The **all** option means that the heading always appears.

Table 1-2. bbs Column Headings

Column Name	Option	Description
SLT	(all)	The slot number
BBNAME	(all)	The name associated with process and slot
QKY	(all)	The message queue key
PID	(all)	The process ID
INS	(all)	The process instance
D	(all)	"YES" if process is a message-sending DIP type; otherwise "NO"
CDATE	(l)	The last process creation time
WK	(l)	The ET work state
SKEY	(l)	The semaphore key associated with process and slot
QID	(l)	The message queue ID
RE-SPA	(l)	The number of respawns from last restart of the voice system
WKCNT	(l)	The ET work count for process

Upon successful completion, **bbs** returns an exit status of zero. Otherwise, **bbs** prints an error message on **stderr** and returns a non-zero exit status if the voice system is not running, or if for some other reason, it can not access the BB.

Example

The following example prints a long listing, displaying all possible fields.

```
bbs -l
```

buildfs

Name

This command builds a speech file system on a specified disk slice.

Synopsis

buildfs -s ***<raw_disk_slice> [-d <dio>] [-h]***

buildfs ***<raw_slice_device> <slice_size> <dio_name>***

Description

The **buildfs** command provides a simple command interface to allow users to build speech file systems on disk slices. **buildfs** operates in two modes. The first mode is identified by the *-s* option as the first argument on the command line. In this mode, **buildfs** does all the administration required to build a speech file system ready for use by the voice file system. In the second mode, which is being supported for historical reasons, **buildfs** operates in the standard fashion, that is it builds a speech file system on *<raw_slice_device>* of size *<slice_size>* with dio name *<dio_name>*. *<raw_slice_device>* is a character device file name such as **/dev/rdsk/0s4**. *<slice_size>* is the size of the speech slice in 16-kbyte blocks. *<slice_size>* must be between 10 and 32767. *<dio_name>* is the name of the disk interface process. Typically, DIO0 is used for speech slices on disk 0 and DIO1 is used for speech slices on disk 1.

Use **buildfs** with the *-s* option to build a speech file system on the specified slice. A slice must be of the form ***D*s*S*** where *D* is the disk number and *S* is the slice number (for example, 0s4). **buildfs** works in the following manner: The slice is checked to see if it is a speech or UNIX file system. If it is a UNIX file system, it is unmounted (if mounted), the **/etc/fsstab** entry is removed, and the **/etc/partitions** file is modified to specify the slice as a raw speech slice. A speech file system is built on the slice, and the particulars about the slice are added to the **/vs/data/fslist** file.

If the slice is not a UNIX file system, an audit runs on the speech file system in an attempt to salvage the file system and preserve the speech on the file system. If the audit exits successfully, the appropriate system files are updated so that when the voice system is started, the speech file system is available. If the audit fails, a speech file system is built on the slice in question and all system files are updated to support the new speech files system.

The *-d* option specifies which dio process to associate with the speech file system. See the above discussion about dio for which dio process to choose.

Example

In the following example, the system creates a 10-block speech file system on slice 0s4 using DIO0.

```
buildfs /dev/rdsk/0s4 10 DIO0
```

Caveats

Note that **buildfs** cannot be used on a disk which has not yet been configured with the operating system. Use **"addspdisk"** to add a new disk to the system. **"addspdisk"** calls **buildfs** as required.

See Also

"addspdisk"

ccarpt

Name

This command generates a call classification data summary report.

Synopsis

ccarpt *<date>*

ccarpt *<start_date> <end_date>*

Description

The **ccarpt** command generates a call classification data summary report. This report is stored in standard out (stdout).

The *<date>, <start_date>*, and *<end_date>* arguments are in the form mm/dd/yy.

Example

The first example generates the call classification data summary report for October 20, 1993. The second example generates the call classification data summary report from October 14 through October 20, 1993.

```
ccarpt 10/20/93
ccarpt 10/14/93 10/20/93
```

cddrpt

Name

This command generates a call data detail report.

Synopsis

cddrpt *<records> <service> <event data> <date>*

Description

The **cddrpt** command generates the call data detail report. This report is stored in standard out (stdout). Before this can be done, the database system must be up and running, but the voice system does not need to be up.

The parameters for the **cddrpt** command are:

- *<records>* — This parameter represents the number of records to be reported. It can be any number, a range of numbers, or “all” indicating all records in the system.
- *<service>* — This parameter represents the script (application) name, or “all” for all applications.
- *<event data>* — This parameter represents a flag indicating whether to include call event data or not. The valid options are either “n” for not including event data or “y” for including event data.
- *<date>* — This parameter is the date the data was collected in the system. The valid options are either a date in mm/dd/yy format or “all” indicating all records in the system.

Example

The following example generates a call data detail report for the first 100 call data collected on date October 20, 1993 for application “balance_chk.” (Call event data if any is also included in the report.)

```
cddrpt 100 balance_chk y all 10/20/93
```

The following example generates a call data detail report for all call data in the system without including call event data.

```
cddrpt all all n all
```

cdsrpt

Name

This command generates a call data summary report for a specific date.

Synopsis

cdsrpt *<hours> <service> <event data> <date>*

Description

The **cdsrpt** command generates the call data summary report for a date specified. The report is stored in standard out (stdout). Before this can be done, the database system must be up and running, but the voice system does not need to be up.

The parameters for the **cdsrpt** command are:

- *<hours>* — This parameter is the hour the call data was collected. It can be any number between 0 to 24 or “all” indicating all 24 hours.
- *<service>* — This parameter is the script (application) name, or “all” indicating all applications.
- *<event data>* — This parameter is a flag indicating whether to include call event data or not. The valid options are either “n” for not including event data or “y” to include event data.
- *<date>* — This parameter is the date the data was collected in the system (in format mm/dd/yy).

Example

The following example generates call data summary report for call data collected between 2 p.m. and 4 p.m. on date October 20, 1993 for all applications on the system. Call event data summary is included in the report.

```
cdsrpt 14-16 all y 10/20/93
```

The following example generates call data summary report for all call data collected on date October 20, 1993 for the application “balance_chk.” Call event data summary is not included in the report.

```
cddrpt all balance_chk n 10/20/93
```

change_device

Name

This command changes the presentation name of a device in the **/vs/data/device_data** file.

Synopsis

/vs/bin/util/change_device

Description

The **/vs/data/device_data** file, as it is initially populated, contains standard generic presentation names (user mnemonic names, see **"add_device"** in this book). The **change_device** command allows a user to change the user mnemonic name from the generic name to a possibly more specific name (for example, a comcode).

The command is interactive and menu driven; the user selects the generic name which they wish to change, then inputs the 2–20 character new name.

Upon the next invocation of the **/vs/bin/configure** program, the new names appear in the device selection menu and in all output generated by the program.

Files

/vs/data/device_data

See Also

"add_device"
"configure"
"get_config"
"remove_device"
"save_config"
"show_devices"
"show_config"

checktf

Name

This command checks for the existence of talkfiles in the voice system.

Synopsis

checktf *<talkfile> [talkfile...]*

Description

The **checktf** command checks for specific talkfiles in the voice system. If the specified talkfile is being used by an existing application **checktf** displays the application name. If no application is associated with the talkfile, **checktf** displays a corresponding message. If the talkfile is currently not being used, **checktf** exits with a value of 0. Otherwise, it exits with the number of talkfiles that were in use from the specified list. Other error exit codes are 199 (incorrect usage) and 200 (voice system is running).

Example

The following example checks for talkfile 41 in the system.

```
checktf 41
```

Warnings

Execute this command only when the voice system is not running.

cmgrtool

Name

This command is a utility that provides trace utilities for the 3270 Protocol Communications Manager (CMGR).

Synopsis

cmgrtool ***[options] [&]***

Description

The **cmgrtool** is a utility that includes a variety of CMGR-related functions, such as collecting binary trace data from the host communications card and writing it to file. This command can be invoked through various switches or interactively.

You can include one or more command line options when starting **cmgrtool** to perform various actions immediately, or you can place **cmgrtool** in the interactive mode and enter commands interactively.

To conserve resources, CMGR does not generate trace data until told to do so. This can be done through **cmgrtool** directives (*t, s, f,* or *l100*) or by supplying the *-d 100* option to the **"load_bin"** command.

The following are valid options for the **cmgrtool** command. Only those options preceded by an asterisk (*) can be used in interactively.

- *-a* — This option specifies that the contents of the internal audit trail (from the communications card) should be continuously output. This activity continues until the UNIX “kill” key is pressed. Diagnostic and informational messages are placed in this audit trail buffer by the CMGR.
- * *-b<cardnumber>* — This option specifies to which card (by number) **cmgrtool** will apply the other switches. The default is 0 (that is, -b0). This option may be combined with other options, although it is only required in multiple card environments.
- * *-i* — This option specifies that the trace facility should ignore any pre-existing trace data in the trace buffer before capturing data to the trace file. This command line option is used in conjunction with *-t, -f*, or *-s* options. Use this option if the event of interest has not occurred yet; do not use this option if the event has already occurred.

- *-k* — This option kills (halts) the communications program, causing program execution on the host communications card to cease; that is, no further responses to host communications will take place. After this command has been issued, the communications card must have its software reloaded in order to resume execution. This option may be used, for example, in a shutting down the UNIX system.
- * *-l<debuglevel>* — This option specifies the CMGR debug level, influencing certain internal CMGR actions. Using this option adds additional overhead to CMGR processes. One of the following *<debuglevel>*s should be provided:

0	This turns off all options.
100	This turns on sync data tracing.

- *-m<filename>* — This option specifies that the content of the data section of the communications card memory is to be dumped to the specified file.

WARNING:
This above option should be used only at the direction of support personnel.

- *-s* — This option specifies the sync-line trace output should be directly displayed on the screen until the UNIX kill key is pressed. This option allows the trace to be viewed in real-time; however, the output is very verbose. In most instances, this option is used in nonmulti-drop sites that are having startup problems.
- * *-t<filename>* — This option specifies that binary sync-line trace data should be collected in the CLEO proprietary form into the specified circular file. The file size maybe specified with the *-T* option. When executing this option, **cmgrtool** is typically placed in the background (using &) and killed after the event of interest has been captured.
- *-T[filesize[blocksize]]* — This option, used with the *-t* option, specifies the maximum size (in bytes, decimal) of the circular file that will be created. When the size limit is reached, new data overwrites the oldest data in the file. The default file size is one Mbyte.
- *-f* — This option displays the version ID of the CMGR program running on the communications card.
- *-h or -?* — This option displays a list of available options with brief descriptions.

When starting **cmgrtool** without specifying any options, a startup message is output to the screen. **cmgrtool** enters interactive mode, displays the interactive prompt (CMGRTOOL>), and waits for commands. The following commands can be used at the **cmgrtool** prompt:

- *a* — This command specifies that the contents of the internal audit trail (from the host communications card) should be continuously output. This activity continues until the UNIX kill key is pressed. Diagnostic and informational messages are placed in this audit trail buffer by the CMGR.
- *d<seg:offset>* — This command displays the host communications card memory contents beginning at the specified address. Issuing subsequent *d* commands without specifying a *<seg:offset>* displays successive memory segments.
- *e<seg:off>* — This command allows you to edit the host communications card memory at the specified address.

  NOTE:
  This is a very advanced operation and should only be performed at the direction of support personnel.
- *-i* — This command specifies that the trace facility should ignore any pre-existing trace data in the trace buffer before capturing data to the trace file. This command can be used in conjunction with the *t*, *f*, or *s* commands. Use this command if the event of interest has not occurred yet; do not use this command if the event has already occurred.
- *k* — This command kills (halts) the communications program, causing program execution on the host communications card to cease; that is, no further responses to host communications will take place. After this command has been issued, the communications card must have its software reloaded in order to resume execution.
- *l<debuglevel>* — This command specifies the CMGR debug level, influencing certain internal CMGR actions. Using this option adds additional overhead to CMGR processes. One of the following *<debuglevel>*s should be provided:

0	This turns off all options.
100	This turns on sync data tracing.
- *m<filename>* — This command specifies that the content of the data section of the host communications card memory is to be dumped to the specified file.

WARNING:
This above option should be used only at the direction of support personnel.

- *s* — This command specifies the sync-line trace output should be directly displayed on the screen until the UNIX kill key is pressed. This option allows the trace to be viewed in real-time; however, the output is very verbose. In most instances, this option is used in nonmulti-drop sites that are having startup problems.
- *t<filename>* — This command specifies that binary sync-line trace data should be collected in the CLEO proprietary form into the specified circular file. The file size may be specified with the *-T* option on the command line when **cmgrtool** is started.
- *q* — This command quits (ends) the interactive cmgrtool session.
- *v* — This command displays the version of the CMGR program running on the host communications card.
- *h* or *?* — These commands display a list of available commands with brief descriptions.

Example

The following example turns on the trace to the file “rawfile.”

```
cmgrtool -t rawfile
```

To turn off the trace specified, enter:

```
kill -9   <pid>
```

where *<pid>* is the process identification number provided when the trace was initiated with **cmgrtool**.

See Also

"dscope"

configure

Name

This command determines allocation of resources for all devices to be included in a VIS system configuration for a given hardware platform.

Synopsis

/vs/bin/util/configure [new]

Description

configure is an interactive command which automatically determines allocation of the following resources within a CONVERSANT VIS system:

- Slot number
- Interrupt
- Direct memory address (DMA) channel
- Input/output (IO) address
- Random access memory (RAM) address
- Serial port number
- Parallel port number

The program is mainly for use by factory personnel for determining the configuration of a system being built for a customer. It is also useful in the field, however, as a tool for field service personnel who are upgrading (adding new devices to) an existing configuration.

All user input is in the form of user friendly prompts or menu selections.

All output from a successful configuration is written to the **/vs/data/conf_data** file. An unsuccessful or incomplete configuration is written to the **/vs/data/fail_data** file so that it may be examined to see why the configuration was not successful. Output to these files is in a compressed format. The **/vs/bin/util/show_config** command is used to view the configurations represented by these files.

The program allows the user to choose the hardware platform they are attempting to configure, followed by all devices which are to be included in the configuration. This ensures that all required devices for the platform are selected.

The program has built-in rules which prevent users from entering devices which are not supported on the hardware platform which they have selected.   It also checks certain hardware feature rules, such as how many of a particular device are supported on the chosen platform. In all cases of device rejection by the program, the user is prompted with a message clearly explaining why the device is being rejected.

Upgrading an Existing Configuration

When **configure** is executed with no argument, it checks to see if a **/vs/data/conf_data** file currently exists. If so, the configuration represented by this file is read and used as the base configuration. This is useful in the case of a field upgrade to an existing VIS. A copy of the **conf_data** file is saved in **/vs/data/conf_MMDDYY** where MM=month, DD=day and YY=year. The existing **conf_data** file is saved because, upon determination of a valid configuration which includes the user's newly specified devices, the current **conf_data** file is replaced by a new one. It may be useful at times to see what the previous configuration looked like. An option on the **/vs/bin/util/show_config** command allows the user to view the configuration represented by the saved conf_MMDDYY file.

Once the file is read, the user is presented a menu of devices they may attempt to add to the current configuration. Once the new devices are specified, the program attempts to allocate resources to the new devices from the pool of resources not currently used by devices already in the configuration. This may be a two pass process:

- PASS 1: An attempt is made to fit the newly specified devices into the current configuration without disturbing any devices currently configured. If this can be done, Pass 2 is not executed.
- PASS 2: If Pass 1 was unsuccessful, Pass 2 attempts the equivalent of a new configuration, unassigning all currently used resources, and pooling the newly selected devices with those already in the configuration. If Pass 2 is successful, the user may be required to change the switch settings on some of the cards already in the system, move them to different slots, etc. If Pass 2 is unsuccessful, the newly specified devices will not fit into the current configuration.

Specifying a New Configuration

If **configure** is executed with no argument and a **/vs/data/conf_data** file does not exist, a menu of currently supported hardware platforms is presented, followed by the menu of devices which may be configured with that platform. The user selects the platform and devices desired, indicating when finished. The program attempts to allocate resources to each device selected. If successful, the program terminates. Otherwise, the user is asked if they wish to remove something and try again.

The **configure** command may be executed with an argument of “new.” This forces a new configuration, as described above, even if a **/vs/data/conf_data** file exists.

Presetting Device Hardware Resources

You may wish to preset certain resources of a single new device being selected for a configuration. It may be desired for example to force the **configure** program to select interrupt 6 for a a particular device being specified. Whenever a single device is specified, the following prompt is shown to the user:

```
Do you wish to preset any hardware options of
device_name? [y|(n)]
```

If “y” is specified, the user is allowed to preset any of the following hardware attributes of the selected device (where applicable):

Interrupt
DMA Channel
IO Address
RAM Address

When the user indicates that they are finished selecting devices, the **configure** program continues as normal. In the case of a new configuration, if a valid configuration is determined, the program terminates normally. If a valid configuration cannot be determined, a message indicating this is displayed and the program terminates. The user is not prompted to remove something and try again if any hardware option for any device was preset.

In the case of an upgrade, the program attempts Pass 1 as described above. If successful, then the program terminates normally. If Pass 1 is unsuccessful, a message indicating this is displayed. Pass 2 is not attempted if the user has preset any hardware options for any device.

Device Data

The devices presented to the user in the menus described above are kept in the **/vs/data/device_data** file. This file is in compressed format. The **/vs/bin/util/"show_devices"** command may be used to view its contents. All attributes of each device are stored in this file.

It may be desirable from time to time to add a new device to those which are configurable in a VIS machine by the **configure** program. The **/vs/bin/util/add_device** prompts a user for all necessary attributes required to add the new device to the device_data file. Once added to the file, the new device is available to the configure program.

Devices which are added to the **device_data** file may also be removed. The **/vs/bin/util/remove_device** program provides this capability.

The names by which devices are presented in the menus described above may be changed to suit a particular user's needs. The **device_data** file is shipped with default generic user names. The **/vs/bin/util/change_device** program allows a user to name devices according to user preference.

Files

/vs/data/device_data
/vs/data/conf_data
/vs/data/fail_data

Notes

Upon reaching the first unresolvable resource conflict, the **configure** program terminates (unless the user selects the **"remove"** option, in the case of a new configuration), and writes the incomplete configuration to **/vs/data/fail_data**. A message indicating which device and which hardware resource caused the conflict is displayed.

The format of the **/vs/data/conf_data** file and the **/vs/data/device_data** file is very specific. A user should not edit or alter these files.

The **"configure"** program should only be run by persons familiar with VIS system configurations and hardware platforms. Refer to Chapter 4, "Running the Configuration Program," in the hardware installation book for your platform for slot position and numbering information. Refer to the documentation on each device for the switch or jumper settings which correspond to the hardware resources determined by the **configure** program.

Serial and parallel port numbering is used for allocation purposes only. There is no correlation between the port number specified in the configuration output and actual physical ports. In configurations where there are multiple ports, any of the available ports may be used for those devices which require one.

See Also

"add_device"
"change_device"
"get_config"
"remove_device"
"save_config"

"show_config"
"show_devices"

copy

Name

This command copies a phrase from the speech database to another UNIX file.

Synopsis

copy phrase *<phrase number>* from talkfile *<talkfile number>* to *<filename>*

Description

The **copy** phrase command copies a phrase from the speech database to UNIX file. The path name for the file may be the full path name or the relative path name. If no path is specified, the file is created in the current working directory. If you are not in the directory in which the phrase to be added is stored, be sure to give the full path name for the talkfile and source file.

NOTE:
Only the login **root** can copy a phrase to any of the root directories.Users without root permission can copy phrases only to directories for which they have permission, usually under their login id.

Example

The following example copies phrase number 2 from talkfile 1 to the file **/speech/talk/a.1**.

```
copy phrase 2 from talkfile 1 to /speech/talk/a.1
```

The following example copies phrase number 174 from talkfile 25 to the file **/speech/talk/h.4**.

```
copy phrase 174 from talkfile 25 to /speech/talk/h.4
```

See Also

"add"
"erase"
"list"

cpuType

Name

This command returns the type of central processing unit (CPU) used in the system.

Synopsis

cpuType

Description

The **cpuType** command returns the type of CPU on the system, either a 386 or a 486. If the **cpuType** command returns a 3, you are using a 386. If the **cpuType** command returns a 4, you are using a 486. To determine the return value, examine the shell variable $?.

cvis_mainmenu

Name

This command accesses the topmost CONVERSANT administrative menus.

Synopsis

cvis_mainmenu

Description

The **cvis_mainmenu** program is a menu interface used to access FACE and the Voice System Administration menus.

See Also

"cvis_menu"

cvis_menu

Name

This command accesses the CONVERSANT VIS Voice System Administration menus.

Synopsis

cvis_menu

Description

This is the command which provides access to the administration of the Conversant Voice Information System. See the *CONVERSANT VIS Version 4.0 Operations*, 585-350-703, for information about administering your VIS.

See Also

"cvis_mainmenu"

dbcheck

Name

This command checks the resources available in the database (Version 6.0 ORACLE).

Synopsis

dbcheck -i

dbcheck r

dbcheck [w n[,m]] [-s] [-e] [-m user[~user...]]

Description

The **dbcheck** command checks spaces usage and rollback segment growth. The **dbcheck** command has three different usages. The *-i* option installs cron entries (optional) to run **dbcheck** at regular intervals and support for logger/alerter messages. (The *-i* option only needs run once). The cron job can be placed in either roots cron file or added to the end of **/vs/bin/util/croncdh** job that runs once a day. This is prompted for interactively. The *-i* option also asks if you want new alerter messages added to the logger/alerter database along with explanations used with the **"explain"** command. This installation only needs to be run if you want the warnings to show up in the system event log or if you want to schedule automatic checking at regular intervals. The *-r* option removes any cron entry set up by the *-i* option.

The third usage actually checks database space against a user set “water marks.” Three different things are checked:

1. Free space
2. Extents against the user set threshold *n* (15% default)
3. Rollback segment(s) growth against the user set threshold *m* (20% default)

When executed, the command generates the appropriate warnings (shown under “Diagnostics” below) if the database falls below *n* percent free or if the rollback segment grows to be more than *m* percent of the total database size.

The command, by default, sends warning messages to the logger/alerter indicating a threshold has been exceeded (the *-i* option must be run first). The -e option disables the entries from going into the log file. The *-s* option prints the warning messages to standard output. The *-m user* option allows for the messages to be mailed to *user*. Multiple users can be sent the mail by separating the user names with ~. Below are sample outputs.

(Output to error log when less than 13% available space/extents or more than 23% used by rollback)

# dbcheck -w13,23

```
*   Mon Feb 15 16:35:06 1993 dbcheck logTest.c:418
DBC001    -- -- ---  Database 10 percent free, 3072 Blocks of 30720 available.
                Reason:  Low DB Space.
*    Mon Feb 15 16:35:06 1993 dbcheck logTest.c:418
DBC002    -- -- ---  Extents low, 100 used of 121, on object MY_TABLE
                Reason:  Low DB Extents
*    Mon Feb 15 16:35:06 1883 dbcheck logTest.c:418
DBC003     -- -- --- Rollback segments=7680 blocks, 25 percent of total space.
                Reason:  High Rollback Usage.
```

Files

LOGROOT=$[LOGROOT:-"/usr/spool/log"}
${LOGROOT}/head/logDBC.h
${LOGROOT}/formats/DBCmsg
${LOGROOT}/formats/formats.mk
${EXPLAINDR}//translateLst
/vs/bin/util/croncdh
/usr/spool/cron/crontabs/root
/usr/spool/cron/crontabs/root.bu

Diagnostics

The **dbcheck** command returns the following values:

0	Success, no limits exceeded
1	Threshold exceeded
2	Processing error
3	Database is not running

Caveat

Once **dbcheck** log messages are installed, using **dbcheck -i**, the alarm priorities, destinations, and thresholds can not be changed through the System Message Display screen as described in Chapter 3, “Configuration Management,” of *CONVERSANT VIS Version 4.0 Operations*, 585-350-703.

See Also

"dbfrag"
"dbfree"
"dbused"
"explain"
"logCat"

## dbfrag

Name

This command lists fragmentation information on the database (Version 6.0 ORACLE).

Synopsis

dbfrag ***[-h -b]***

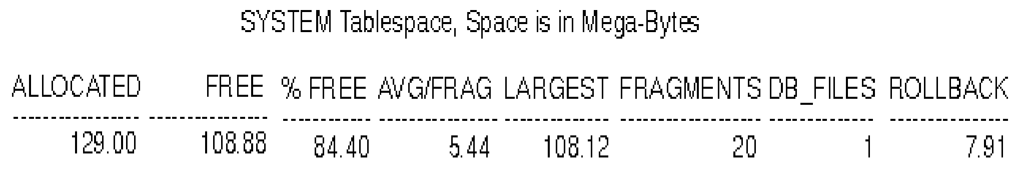
Description

The **dbfrag** command is a shell script that reports on database allocation, usage, and fragmentation. The block size reported is in ORACLE blocks (2048 bytes). You can request the information to be reported in mbytes with the *-b* option. This tool is useful to get a quick check on database usage and provides a shell interface into some key ORACLE statistics.

This tool only reports on information in the ‘SYSTEM’ tablespace. With the *-h* option, the listing will be printed without a header. This option is useful if you want to parse this output to get select a specific field.

The following requests fragmentation information in Mbytes (using the *-b* option).

# dbfrag -b



```
                    SYSTEM Tablespace, Space is in Mega-Bytes

ALLOCATED        FREE  % FREE  AVG/FRAG  LARGEST  FRAGMENTS DB_FILES  ROLLBACK
---------------- ---------------- ------------ ---------------- ------------- ------------------- -------------- ----------------
   129.00      108.88   84.40      5.44   108.12         20        1      7.91
```

Examples

The following example gets the largest contiguous ORACLE space available.

dbfrag -h|awk ‘length>1 {print $5}’

```
10240
```

Diagnostics

The program returns the following:

0	Success
1	Processing Error

See Also

"dbcheck"
"dbfree"
"dbused"

dbfree

Name

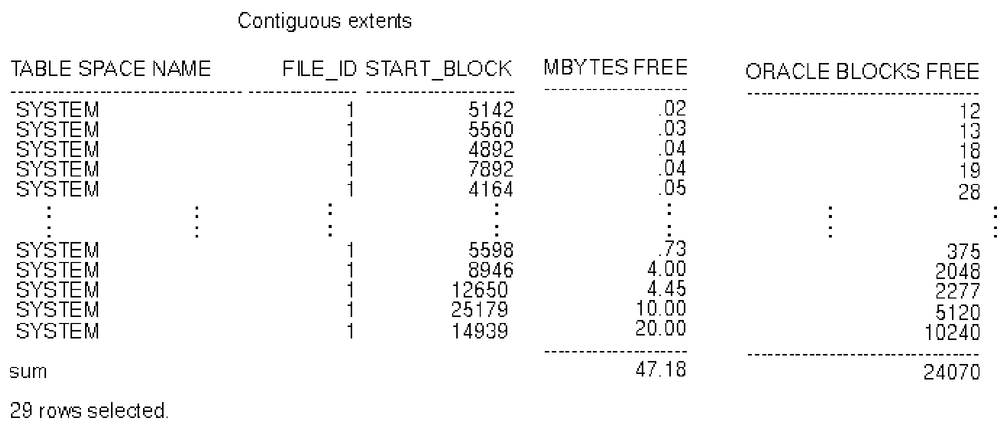
This command checks the space available in the database by partition (Version 6.0 ORACLE).

Synopsis

dbfree ***[h]***

Description

The **dbfree** command is a shell script that lists the amount of free space in the database by free contiguous blocks. The result is a detailed listing of each free memory area followed by the sum of each partition. The free blocks are listed in 2048 bytes/block (ORACLE blocks). There is also a column that lists the same information in mbytes. The *-h* option removes the column headers. Below is a sample output of the **dbfree** command.



```
                         Contiguous extents

TABLE SPACE NAME              FILE_ID START_BLOCK   MBYTES FREE    ORACLE BLOCKS FREE
----------------------------- ------- -----------   -----------    ------------------
SYSTEM                              1        5142           .02                    12
SYSTEM                              1        5560           .03                    13
SYSTEM                              1        4892           .04                    18
SYSTEM                              1        7892           .04                    19
SYSTEM                              1        4164           .05                    28
  :                 :           :              :             :         :            :
  :                 :           :              :             :         :            :
SYSTEM                              1        5598           .73                   375
SYSTEM                              1        8946          4.00                  2048
SYSTEM                              1       12650          4.45                  2277
SYSTEM                              1       25179         10.00                  5120
SYSTEM                              1       14939         20.00                 10240
                                                    -----------    ------------------
sum                                                       47.18                 24070

29 rows selected.
```

Diagnostics

The program returns the following values:

0 Success

1 Processing Error

Caveats

The **dbfree** command creates a temporary table “dba_fragments” under user system that compresses the adjacent entries provided by the dictionary view “dba_free_space.”

See Also

"dbfrag"
"dbcheck"
"dbused"

## dbused

Name

This command provides database use by oracle user (Version 6.0 ORACLE).

Synopsis

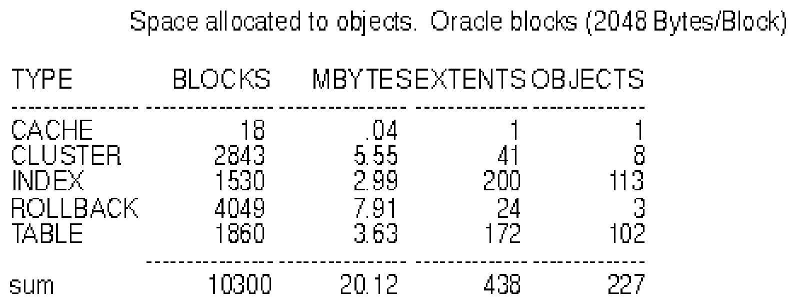
dbused ***[hs] [u <uid/passwd>]***

Description

The **dbused** command is a shell script that shows the amount of space used by each object for a given user. Objects are tables, indexes, clusters, rollback, and cache. The default user is sti/sti. The *-s* option reports summary information grouped by objects. The special user “all” reports information for the entire database. The *-h* option skips the header message. This option is useful if you are parsing. The *-u <uid/passwd>* option allows the user to specify the oracle user id and password (the default is sti/sti, all for all users).

Below is an output summary for user “all.”

# dbused -su all

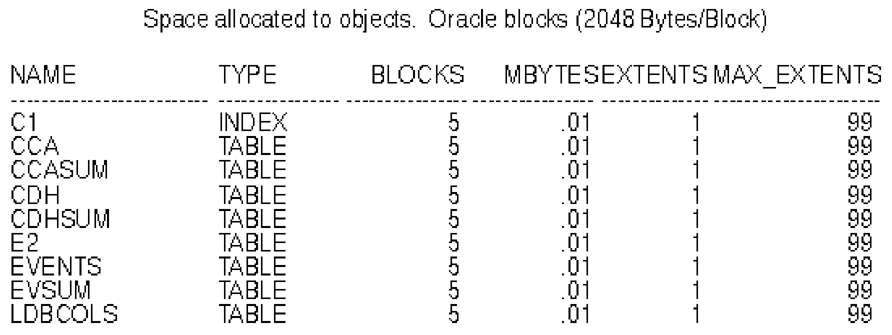


```
              Space allocated to objects.  Oracle blocks (2048 Bytes/Block)

TYPE            BLOCKS      MBYTESEXTENTSOBJECTS
---------------- --------------- --------------- ------------- -------------
CACHE               18          .04          1         1
CLUSTER           2843         5.55         41         8
INDEX             1530         2.99        200       113
ROLLBACK          4049         7.91         24         3
TABLE             1860         3.63        172       102
                 --------------- --------------- ------------- -------------
sum              10300        20.12        438       227
```

Below is output for user “sti.”

# dbused



```
                    Space allocated to objects.  Oracle blocks (2048 Bytes/Block)

NAME                TYPE            BLOCKS     MBYTESEXTENTS MAX_EXTENTS
------------------- --------------- --------------- --------------- ------------- ---------------------
C1                  INDEX                5       .01       1          99
CCA                 TABLE                5       .01       1          99
CCASUM              TABLE                5       .01       1          99
CDH                 TABLE                5       .01       1          99
CDHSUM              TABLE                5       .01       1          99
E2                  TABLE                5       .01       1          99
EVENTS              TABLE                5       .01       1          99
EVSUM               TABLE                5       .01       1          99
LDBCOLS             TABLE                5       .01       1          99
```

Diagnostics

The program returns the following values:

0	Success
1	Processing Error

See Also

"dbfrag"
"dbfree"
"dbcheck"

delete card/channel

Name

This command removes a card or channel from a service or an equipment group.

Synopsis

delete card *<chan.[port]>* from *[eqpgrp] <group number>*

delete channel *<number>* from *[eqpgrp] <group number>*

Description

Delete card/channel removes the specified card or channel from a service or equipment group. The parameters for the delete card/channel command are:

- *<chan.[port]>* — This option specifies the card/channel number (a single card/channel number from a range of 0–255, a range of card/channel numbers in the form m–n, or the word “all” for all card/channel numbers).
- *eqpgrp* — This option specifies the “svcgrp” when deleting from a service group or “eqpgrp” when deleting from an equipment group. If no group type is given, the “svcgrp” is assumed.
- *<group number>* — This option identifies the equipment group or service group.

If you want to remove all cards or channels from a equipment group, it may be easier to delete the entire equipment group than to delete channels or cards. To delete an equipment group, use the **"delete eqpgrp"** command.

Example

The following example deletes card 4 from service group 1.

```
delete card 4 from svcgrp 1
```

The following example deletes channels 10 through 13 from equipment group 3.

```
delete channel 10-13 from eqpgrp 3
```

See Also

"audit"
"delete eqpgrp"
"delete service"

delete eqpgrp

Name

This command removes an equipment group.

Synopsis

delete eqpgrp *<group number>*

Description

The **delete eqpgrp** removes an equipment group. The *<group number>* argument is the equipment group list. To remove all equipment groups, use the word “all” as the group number.

Example

The following example removes equipment group number 3.

delete eqpgrp 3

The following example removes all equipment groups.

delete eqpgrp all

See Also

"assign service"
"audit"

delete service

Name

This command removes the telephone number for a script from a group.

Synopsis

delete service *[application_name]* from *<chan|dnis> <chan number | phone number>*

Description

Delete service removes from a group the telephone number assigned to a script. The parameters for the **delete service** command are:

- *application name* — This optional parameter specifies the name of application.
- *<chan | dnis>* — This parameter specifies the name of the service group.
- *<chan number | phone number>* — This parameter contains a list of one or more channels or telephone numbers separated by blanks. The word “any” or “all” shows that service is removed from all calls regardless of what number was dialed.

Only telephone numbers that have been assigned using the **"assign service"** command can be deleted.

Example

The following example removes the application cashmgmt1 from service on channel group 1 with channel number 4523.

```
delete service cashmgmt1 from 1 4523
```

The following example removes service group 1 from channel number 8922.

```
delete svcgrp 1 8922
```

See Also

"assign service"
"audit"

detach

Name

This command places a unit in the nonexistent state.

Synopsis

detach *<unit> <number> [-i] [-n]*

Description

The **detach** command places a unit currently in the manual-out-of-service (MANOOS) state into the nonexistent (NONEX) state. Before this can be done, the unit must be taken from the in-service (INSERV) or broken (BROKEN) state and put in the MANOOS state using the **"remove"** command.

The parameters for the **detach** command are:

- *<unit>* — This parameter identifies the unit. The choices are “channel” or “card.”
- *<number>* — This parameter specifies the channel or card number, a range of channel or card numbers in the form m–n, or the word “all” for all the channel or card numbers. Card numbers are in the form ***card#[.port#]*** where *port#* is the port of the card *card#*. If *port#* is not given, all ports of the card specified are detached. If no card number or channel is given, a syntax message is displayed.
- *-n* — This optional parameter disables prompting from the system whether to wait until a conflict has been resolved (see the *-i* option below) or to terminate the request to detach.
- *-i* — This optional parameter is used to enable secondary command registration. If T1 diagnostics are being run, this option allows the “detaching” of another card. If *-i* is used and another maintenance command is being run (**"remove", "detach", "attach", "restore", diagnose**), the request to **"detach"** is blocked and a message is printed to the screen. If *-i* is not used and any maintenance command is being run, the request to **"detach"** is blocked and a message is printed to the screen.

  If the command is permitted to run, a check is made to see if the command is in conflict with another. A command is in conflict if the card or card associated with it:

  1. Is the T1 card being diagnosed
  2. Will cause a change in the existing TDM bus master assignment

3. Has an interdependency with the T1 card being diagnosed (for example, PRI)

If one of the above conflicts exist and *-n* is not used, the user is asked whether to wait until the conflict is resolved or to terminate the request. If T1 diagnostics are executing on-line tests and a conflict is detected, the **detach** command is blocked. If T1 diagnostics are executing off-line tests and a conflict is detected, the user is asked whether to wait until the conflict is resolved or to terminate the request to detach.

To delete out of the command, press DEL. If this does not stop the command, you may need to press CTRL and backslash simultaneously. If, while running **detach**, you wish to abort the command, a message similar to the following may appear:

```
At the user’s request, administration of the following
cmd(s) has been interrupted.

CARD NUMBERS: <card numbers>

To assure proper operation of the identified card(s),
run diagnostics at the earliest opportunity.
```

It is recommended when **detach** is aborted, diagnostics be run on all cards being administered to ensure they are returned to a fully functional state.

Example

The following example detaches card 4 and places it in the nonexistent state as far as the system is concerned.

detach card 4

The following example detaches channels 1 through 3 and places them in the nonexistent state as far as the system is concerned.

detach channel 1-3

See Also

"attach"
"remove"
"restore"

diagnose bus

Name

This command tests a bus while it is in service.

Synopsis

diagnose bus *<bus number>*

Description

The **diagnose bus** command tests a bus while it is in service. The *<bus number>* option is the number of the bus you want to diagnose. The valid option for the number argument is 1. If the *immed* option is used, any calls currently being processed are dropped immediately.

This command changes the temporary state of a unit to diagnostic (DIAG). If a unit fails the diagnostics, the permanent state is changed to BROKEN; otherwise, the permanent state is unchanged.

If the bus passes diagnostics, a message such as the following is displayed:

```
Diag TDM n: Diagnostics completed.
```

You may also receive a message like the following:

```
Diag TDM n: Diagnostics bus n failed.
```

To delete out of this command, press DEL. If this does not stop the command, you may need to press CTRL and backslash simultaneously.

Example

The following example diagnoses bus 1.

diagnose bus 1

diagnose card

Name

This command tests a card while it is in service.

Synopsis

diagnose card *<card number> [option]...*

Description

The **diagnose card** command is done at the card level for any card in the system. The *<card number>* option is the number of the cards you want to diagnose. The word “all” can be used to specify all cards.

This command changes the temporary state of a unit to diagnostic (DIAG). If a card is stuck in the INSERV state, use the **diagnose card <number> immed** command. This temporarily removes the unit from the busy state unconditionally and places it in the manual-out-of-service (MANOOS). Note that any calls on the card when the “immed” option is used are dropped immediately.

For T1 cards the valid options are:

- *-n* — This option prevents prompting from the system during diagnostic tests. The diagnostics assume the default values during the test and the user is informed when the diagnostics are completed.
- *-i* — This option is used to enable secondary command registration.See the description of *-i* for T/R and SP cards below.

For T/R and SP cards, the valid options are:

- *-n* — This option disables prompting from the system whether to wait until a conflict has been resolved (see the *-i* option for T/R and SP cards below) or to terminate the request to diagnose.
- *-i* — This option is used to enable secondary command registration. If T1 diagnostics are being run, this option allows the diagnose of another card to be performed. If *-i* is used and another maintenance command is being run (**"remove", "detach", "attach", "restore"**), the request to diagnose a non-T1 card is blocked and a message printed to the screen. If *-i* is not used and any maintenance command is being run, the request to **diagnose card** is blocked and a message printed to the screen.

If the command is permitted to run, a check is made to see if the command is in conflict with another. A command is in conflict if the card or card associated with it:

1. Is the T1 card being diagnosed
2. Will cause a change in the existing TDM bus master assignment
3. Has an interdependency with the T1 card being diagnosed (for example, PRI)

If one of the above conflicts exist and *-n* is not used, the user is asked whether to wait until the conflict is resolved or to terminate the request. If T1 diagnostics are executing on-line tests and a conflict is detected, the **diagnose card** command is blocked. If T1 diagnostics are executing off-line tests and a conflict is detected, the user is asked whether to wait until the conflict is resolved or to terminate the request to diagnose.

If a unit fails the diagnostics, the permanent state is changed to BROKEN. If the unit being diagnosed previously was marked BROKEN and it passes diagnostics, it is put in the MANOOS state. Otherwise, the permanent state is unchanged.

When diagnostics are complete, T1 and SP cards are reinitialized and the appropriate software is downloaded to the cards.

For T/R cards, additional diagnostics first check the hardware status of the card specified. Then the system tests for dial tone on the card's channels not in the NONEX state. The result of the dial tone test is one of the following:

1. Nonex state — This channel is not checked for dial tone.
2. Dial Tone Found — This channel is operational.
3. NO Loop Current — The hardware cannot find telephone loop-current from the Central Office, PBX, or ACD. There probably is no phone line on this port.
4. NO Dial Tone — The hardware has found telephone loop-current but could not detect dial tone on this port.

If at least one channel detects dial tone, the entire card detects these frequencies as dial tone. If no channels detect dial tone, the card defaults to 330 and 440 Hz. The outcome of the dial tone tests do not affect the pass or fail results of the diagnostics. If no loop current is detected on a channel but the channel passed diagnostics, the channel is placed in MANOOS. In this case, the card does not become IDLE regardless of its previous state.

If a T/R and/or SP card passes diagnostics, a message such as the following is displayed:

```
Diagnose <card> n, Passed.
```

If a T1 card passes diagnostics, a message such as the following is displayed:

```
All tests passed.
```

You may also receive a message for a T/R and/or SP card saying:

```
Diagnose <card> n, failed <reason>
```

If a T1 card fails diagnostics, a help screen is provided giving you information to help resolve the reason for the failure. If you try to diagnose cards that are not installed in the system or if they are installed but are in the nonexistent state, an error message is displayed.

To delete out of the command, press DEL. If this does not stop the command, you may need to press CTRL and backslash simultaneously. Be aware, however, that this fixes the console, but does not terminate the diagnostic routine. If, while running diagnose, you wish to abort the command, a message similar to the following may appear:

```
At the user’s request, administration of the following
cmd(s)has been interrupted.

CARD NUMBERS: <card numbers>

To assure proper operation of the identified card(s),
run diagnostics at the earliest opportunity.
```

It is recommended when **diagnose** is aborted, diagnostics be run again on all cards being administered to ensure they are returned to a fully functional state.

Examples

The following example runs diagnostics on card number 3.

diagnose card 3

The following example runs diagnostics on cards 4 through 7.

diagnose card 4-7

This example runs diagnostics on cards 4 through 7 immediately, dropping all calls currently in progress.

diagnose card 4-7 immed

dip_int

Name

This command sends DIP interrupt to a script on a channel or a range of channels.

Synopsis

dip_int *<channel>*

dip_int *<channelStart-channelEnd>*

Description

The **dip_int** command sends a message or messages to TSM requesting that TSM send interrupt messages to the script running on *<channel>* or the range of channels *<channelStart-channelEnd>*. If no script is running on the channel or if TSM does not own the channel, no action is taken for the channel. **dip_int** does not wait for a response from TSM.

Scripts running on the channel receive the EDIPINT event.

CAUTION:
Be careful when you use this command. It may affect other applications running on the system.

Example

The following example requests that TSM send interrupt messages to channel two.

dip_int 2

The following examples requests that TSM send interrupt messages on channels one through 32.

dip_int 1-32

Return Values

If the **dip_int** is successful, a 0 value is returned. If any value other than 0 is returned, the **dip_int** command completely or partially failed.

If **dip_int** returns a value of 2, then **dip_int** failed due to temporary condition. In this case, the user should attempt the **dip_int** command again.

See Also

"soft_disc"

display card

Name

This command displays information about specified cards.

Synopsis

disp card *<option> [option]*

Description

The display card command displays data about a specified card or about cards in a specified state. Use the *|lp* option to send the output to the line printer.

The display card command options are:

- *<card#>[.port#]* — This option displays information on card *<card#>* and on port *<port#>* of the specified card. All ports are shown if *<port#>* is not given. A range of cards may be specified in the form m–n without using the *<port#>* option.
- *all* — This option displays information on all cards.
- *mtc* — This option displays all cards being used by the maintenance process.
- *tsm* — This option displays all cards being used by the TSM process.
- *tr* — This option displays all Tip/Ring (T/R) cards.
- *manoos/moos* — This option displays all cards in the out-of-service state.
- *nonex* — This option displays all cards in the nonexistent state.
- *broken* — This option displays all cards in the broken state.
- *inserv* — This option displays all cards that have at least one channel in the in-service state.

If more than one option is used, only cards that satisfy all the options given are displayed. If an invalid combination of options is given, an error message is displayed.

Example

The following example displays card information on channel 2 port 0.

```
disp card 2.0
```

The following example displays information on all cards.

```
disp card all
```

The following example displays information on all cards in the state “Mtc.”

```
disp card mtc
```

The following example displays information on all tip/ring (T/R) cards in the state “Broken.”

```
disp card tr broken
```

display channel

Name

This command displays channel information.

Synopsis

disp channel *<option> [option]*

disp chan *<option> [option]*

Description

This command is used to list information at the channel level. The **display channel** command options are:

- *chan number* — This option displays information on the channel specified by *chan number.* A range of channels may be specified in the form m–n.
- *all* — This option displays information on all channels.
- *mtc* — This option displays all channels being used by the maintenance process.
- *tsm* — This option displays all channels being used by the TSM process.
- *svcgrp <group number>* — This option displays channels assigned to service group *<group number>*. A syntax message is displayed if no group number is given.
- *telephone <tel number>* — This option displays channels with telephone numbers assigned.
- *tr* — This option displays all Tip/Ring (T/R) channels.
- *manoos/moos* — This option displays all cards in the out-of-service state.
- *nonex* — This option displays all cards in the nonexistent state.
- *broken* — This option displays all cards in the broken state.

If more than one option is used, only channels that satisfy all the options given are displayed. If an invalid combination of options is given, an error message is displayed.

Example

The following example displays information for channel 1.

```
disp channel 1
```

The following example displays information all channels being used by the TSM process.

```
disp channel tsm
```

The following example display information on all channels.

```
disp channel all
```

display eqpgrp

Name

This command displays an equipment group report.

Synopsis

disp eqpgrp ***<group number>***

Description

The **display eqpgrp** command is used to list all the equipment assigned to the specified equipment group. The *<group number>* is the number of the equipment group. If the group number is missing invalid, a syntax message is displayed. If you specify “all,” every equipment group is displayed.

Example

The following example lists all the equipment assigned to equipment group 1.

disp eqpgrp 1

The following example lists all the equipment assigned to equipment groups 2 through 20.

disp eqpgrp 2-20

The following example lists all equipment assigned to all equipment groups.

disp eqpgrp all

display messages

Name

This command displays system (error) messages.

Synopsis

display messages [-c]

> ***[priority <alarms,critical,'*C',major,'**',minor,'*',events,all>]***
> ***[start <mm/dd HH:MM:SS>]***
> ***[stop <mm/dd HH:MM:SS>]***
> ***[card <range,T1,TR,SP,...,all>]***
> ***[channel <range,T1,TR,SP,...,all>]***
> ***[ID <message ID1,message ID2,all>]***
> ***[source <TSM,VROP,SPIP,TRIP,...,all>]***
> ***[pattern <regular expression search pattern>]***
> ***[number,all]***

Description

The **display messages** command displays error and status messages which have been logged by the voice system. Various options are provided so that the display can be limited to specific types of messages. If no arguments are supplied to **display messages**, information is displayed on how to read the messages (the message format) as well as command usage. The messages are written to standard output.

If there are too many messages to be displayed on the screen, only enough will be displayed to fill the screen. You will be prompted with "Press the ENTER key to see more, or enter 'q' to quit." If you do not wish to be prompted to press ENTER (that is, display all of the messages at once), you may use the *-c* option.

To display all of the system messages that have been logged, type **display messages all**. Display the last 100 messages by replacing the *all* argument with '100', that is by typing **display messages 100**. The number or *all* argument need not be used if one of the other options is also being used. If it is omitted, it defaults to *all*.

If you wish to display only specific types of messages, you may precede the number of messages to be displayed with one or more of the options *priority, source, start, stop, id, card,* or *channel*. If more than one of the options is specified, only messages that meet all of the specifications are displayed.

The *priority* option can be used to display messages with specific types of urgencies. There are two groups of priorities, alarms and events. Alarms are all those messages that have been reported as *C (critical), ** (major), or * (minor) priorities. Events are all the remaining messages that have no priority (for example, status messages). For example, to display the last 100 alarms, type **display messages priority alarms 100**.

You can also display specific priorities using the *priority* option. You can specify either the name of the priority or its symbol (for example, critical or *C) For example, to display all of the critical messages type **display messages priority critical all**. Combinations of priorities can also be displayed by listing each priority separated by a comma. For example, display the last 100 alarms messages by typing **display messages priority '*C','**','*' all** where *C, **, and * must be enclosed in quotes.

The *start* option allows you to specify a starting time for display of messages. Only messages that were logged on or after the time you specify will be displayed. The time can be specified by date and/or a time. You can specify the date in a variety of ways: "May 1, 1992", "05/01/93", and "05-01-93" are all valid dates. The word "today" is equivalent to specifying the current date. The time can be specified in several ways: hh:mm:ss, hour=hh, min=mm, sec=ss, where hh is 0 to 23, and mm and ss can be 0 to 59. DO NOT mix the hh:mm:ss format with the item==xx format. If portions of the time are not specified, they default to 0 hours, 0 minutes, and 0 seconds, that is, giving only the time of day indicates today's date. For example, if today is January 15, 1993, **display messages start "12/31 09:00"** displays all of the messages that were logged starting at 9 am on December 31, 1993. In order to display messages from a previous year, you *must* specify the year. The entire start date and time must be enclosed in quotes (for example, **display messages start "April 21, 1993 13:00:00"**). If only the date is specified, the time defaults to the beginning of the day. So, for example, **display messages start today** displays all of the messages that were logged today (the day in which the command is being executed).

The *stop* option allows you to display messages logged up to a specific time. The date and time syntax is the same as that for the *start* option. Therefore, **display messages stop today** displays all messages that were logged before today.

The *start* and *stop* options can be used together to display messages that were logged over a specific period of time. For example, **display messages start "May 1" stop "May 2"** displays all messages logged on May 1 of this year.

NOTE:
If you want the start and stop options to be the same day (for example, May 1), you must specify the hours and minutes for which you want to display messages. Otherwise, the time defaults to 00:00 for both the start and stop options and no messages are displayed.

The *card* option allows you to specify messages logged about a specific card or cards. For example, **display messages card 2** displays all messages logged that are associated with card 2. You can display combinations of cards. For example, **display messages card 2,3** displays messages for cards 2 and 3. **display messages card 0-2** displays messages for cards 0, 1, and 2.

You can also use the *card* option to display messages logged about a specific type of card. For example, **display messages card t1** displays all messages logged about T1 cards.

The *channel* option works in the same fashion as the *card* option. For example, **display messages channel tr** displays all messages logged about Tip/Ring (TR) channels. **display messages channel 5** displays all messages logged about channel 5. (Note that the *channel* option requires an argument. Typing **display messages channel 100** attempts to display all messages pertaining to channel 100. If you want to display the last 100 messages pertaining to any channel, type **display messages channel all 100**.)

Note that specifying both the *card* option and the *channel* option displays all of the specified card related messages but of the channels that are specified only those that reside on the specified cards are displayed. For example, **display messages card t1 channel all 100** displays the last 100 messages logged for T1 cards and T1 channels. **display messages card t1 channel tr** never displays any messages as it is impossible for a TR channel to reside on a T1 card.

The *source* option allows you to display messages logged by a particulars system process. For example, some of the standard system processes are:

Process Name	Function	Types of Messages Reported
ASAI	Adjunct/Switch Application Process	ASAI Problems/Status
MTC	System Maintenance Process	Card/Channel status, Diagnostic Results
SPIP	SP Card Interface Process	Speech,TTS,PRI,SR Problems/Status
TSM	Script interpreter/processor	Script Problems
TRIP	Tip/Ring (Analog) Interface Process	TR Problems/Status
TWIP	T1 Interface Process	T1 Problems/Status
VROP	Speech Database Process	Playback/Coding Database Problems

For example, **display messages source TRIP** displays all messages logged regarding T/R cards and channels.

The *pattern* option allows you to specify a regular expression as accepted by **"logCat"** that may appear in any part of a message. (Refer to **"logCat"** later in this book for additional information.) The *pattern* must enclosed in quotes and surrounded by slashes (/). For example, **display messages pattern '/XYZ/'** provides all messages that use the pattern XYZ anywhere in the message. Note that this option is case-sensitive.

The *id* option allows you to display specific message ids that have been logged. For example, **display messages id TWIP004** displays all occurrences of that message. For example, **display messages id TWIP004,TWIP009** displays all occurrences of both messages.

The *number* option specifies the number of messages you want to display, or use the *all* value to display all messages. The command excepts a three digit number so you may display up to 999 messages.

Display Format

All messages are displayed with two or three lines of information. Messages are separated by a blank line to ease viewing. Each message displayed conforms to the format shown in following figure.

```
PR DAY MON DD HH:MM:SS ZZZ YYYY   SOURCE
TTTTTTTT YY UU NUM TEXT...
TEXT (Continuation if necessary.)
blank line
```

Field	Meaning	Examples
`PR`	Priority	`*C` (Critical), `**` (Major), `*` (Minor), “ “(Event)
`DAY`	Day	`Sun - Sat`
`MON DD`	Date	`Jan 1 - Dec 31`
`HH:MM:SS`	Time	`00:00:00 - 11:59:59`
`ZZZ`	Time Zone	`EST, EDT, CST...`
`YYYY`	Year	`1992,...`
`SOURCE`	Source	`TSM, TWIP, VROP,...`
`TTTTTTTT`	8 char Msg ID (Tag)	`TWIP2104,...`
`YY`	FRU Type	`TR,T1,SP`, or `HO` or `--` if N/A
`UU`	Unit Type	`CA` (Card) or `CH` (Channel) or `--` if N/A
`NUM`	Unit Number	`000` to `999` or `---` if N/A
`TEXT`	Message Text	Varies with message (See below)

Figure 1-1. System Message Display Format

The **TEXT** field can be one or two lines long.

Example

The following example is representative of the output from typing **display messages**:

```
                  MESSAGE LOG REPORT

Pr    Time                                      Source
--    ----                                      ------

** Wed Dec 30 15:55:16 1992                     TWIP
   TWIP017  T1 CA   0   Facility out of service.
            Reason: Blue alarm

*  Wed Jan  6 13:38:21 1993                     TRIP
   TRIP002  TR CA   1   Corrupted data detected on TDM bus.
            Timeslot 254. Reason: TDM Parity Error

*  Wed Jan  6 13:41:52 1993                     TRIP
   TRIP005  TR CH  24   No loop current.
```

See “Description” for this command for more examples of command usage.

display service

Name

This command lists all valid services or scripts.

Synopsis

display service

disp service

Description

The **display service** command lists all valid services, or scripts, on a system.

Example

The following examples lists all valid services or scripts currently on the system.

```
disp service
```

displaypkg

Name

This command lists the software packages installed on the VIS.

Synopsis

displaypkg

Description

The **displaypkg** command lists the software packages currently installed on the VIS. It is helpful to use this command to see what software is on the system before using either the **"installpkg"** or **"removepkg"** commands.

Example

The following example lists all the software packages currently installed on the VIS.

```
displaypkg
```

See Also

"installpkg"
"removepkg"

dscope

Name

This command is a utility that displays the trace data collected by **"cmgrtool"**.

Synopsis

dscope *[options] <filename> [> output file]*

Description

The **dscope** command is an off-line utility that displays the trace data collected by **"cmgrtool"**, formatted in a protocol-specific data scope fashion for ready analysis.

After you have collected sufficient sync-line traffic, kill the **"cmgrtool"** process. This causes it to flush its buffers and close the trace file. You may find it convenient to run **dscope** a number of times against the same trace file, each time specifying different options. This allows you to view varying degrees of detail in the trace information. Generally, the initial pass should include the *-c* and *-s=100* options to minimize the volume of the output. The specific problem you are looking to resolve via **dscope** dictates what the appropriate combination of options are. Many of the available options influence the amount and type of detail that **dscope** displays.

The following options are used for the **dscope** command:

- *-c* — This option causes the trace listing to be compressed by showing only the first line of each sync-line data frame and not putting a blank line between frames. Use this mode to quickly view the overall trace, rather than the detail. This is useful for quickly locating areas of trace data that should be examined in detail.
- *-d* — This option causes the delta time (the elapsed time between the trailing edge of the previous frame and the trailing edge of the current frame) to be displayed with each frame. This value (in units of milliseconds) represents the sum of the actual send/receive time of the current frame and any idle or turn-around time that preceded this frame. It appears at the end of each block of frame data. The default is to not display the delta time. This option cannot be combined with the *-t* timestamp option.

- *-i* — This option causes the Receiver Ready (RR) frames to be included in the output. This option is primarily used to check poll/final bit. The default is that no RR frames are output. This option causes very verbose output (see the *-s* option alternative).
- -*l=lines* — This option specifies the number of line per page. Specifying zero suppresses the generation of page header (recommended when output is piped into more). The default is 66.
- *-n* — This option causes the SDLC NR/NS (receive/send) frame counts, as well as the Poll/final bit, to be displayed. This option only applies to SNA traces. It may be useful in situations where excessive retransmission of frames is a problem. The default is to not display this information.
- *-p=<addr>* — This option specifies that **dscope** should output trace data for a specific PU (or control unit) address only. *<addr>* is specified in decimal format.

  Because tracing always includes all control unit addresses in BSC environments, this option is most useful in multi-drop BSC lines. This option is rarely needed in SNA/SDLC environments because tracing becomes limited to your configured SDLC/PC address after a SNRM has been received. Traffic to other addresses (on a multi-drop line) will no longer be traced after the host starts talking to you. Tracing will automatically expand to include all addresses should the host subsequently issue a DISC command (or stop transmitting to your address for 60 consecutive seconds).
- *-r* — This option reverses the transmit data (TX) and the receive data (RX) definitions. This option is used only on trace file collected on the DCE port of the Feline Datascope product.
- *-s=<rrcount>* — This option specifies that Receiver Ready (RR/RNR) counters should be shown when the number of consecutive RRs exceeds *<rrcount>* between data frames (only the number of RRs transmitted and received is displayed). A sufficiently large *<rrcount>* value (50–200) will result in an entry in the trace output that documents the occurrence of the line idle time. A smaller *<rrcount>* value shortens the required idle time to generate an output record, while larger values do the opposite and usually output fewer occurrences. Use the *-i* option to see every RR/RNR.
- *-t* — This option causes the time stamp to be displayed on each frame, appearing at the end of each block of frame data. The default is to not display the time stamp. This value (in milliseconds) is relative to when the 3270 software was downloaded to the host communications card (not the current clock time) and wraps back to zero every 49 days. This option cannot be combined with the *-d* (delta time) option.
- *-u* — This option causes **dscope** to output a **utep** compatible input file, used by support personnel to reproduce the a customer scenario.

- *-2=<width>* — This option specifies the width for printed output in number of columns. The default value is 80. This value influences the amount of viewed data when combined with the *-c* option. Try using **-w=132 -c** for wide-carriage (or compressed or landscape) printers.
- *-x* — This option creates expanded trace output. All non-data entries (such as modem signals) from the native trace file are also output. This option produces very verbose output and primarily used to solve startup problems.
- *-h* or *-?* — This option displays all available options.
- *<filename>* — This is the name of the trace file created by **"cmgrtool"** to be read by **dscope**.
- *[> <outfile>]* — This specifies the file or device to which the output from **dscope** is to be written. For large traces, it may be convenient to redirect **dscope** output to a **/tmp** file and examine this file with your favorite editor rather than printing.

Example

The following example converts the trace file “rawfile” into a readable file “datafile.”

```
dscope rawfile > datafile
```

See Also

"cmgrtool"

edExplain

Name

This command edits the explanation text for one or more message tags.

Synopsis

edExplain ***{msgID} [...]***

Description

The **edExplain** command edits the explanation text for one or more message tags.

The following are environment variables for the **edExplain** command:

EDITOR	The program used to "edit" the explanation text. Default: vi
EXPLAINDIR	The root directory of the explanation texts. Default: **/gendb/data/explain**
VERBOSITY	If set to anything, **edExplain** will run verbosely.

An explanation file is basically a clear text file. Its contents are displayed "as is" to the user when this explanation is requested. If it is a primary explanation procedure (an explanation that the end user will want to reference by name), it should begin with a line of the form:

<< {tag} [{tag}...] >>

which identifies the explanation or procedure and all its alternate names as defined in the translation file, **$EXPLAINDIR/translateLst**.

There are two exceptions to the rule that the file contains clear text that will be displayed to the user:

1. Any line beginning with a "#" character is considered to be an internal comment and is not displayed
2. Lines beginning with ".explain" are special directives to include at this point another explanation text in place of this line.

Example

In the following example, the first line is the SCCS identification line and is not displayed to the end user. The second line identifies the explanation. Then the text describing the problem follows.

```
#%W% %T% %H%
<< TWIP007 TWIP_BDERR >>
.... text of explanation describing what a T1 card error means...
```

erase

Name

This command deletes a phrase from the speech database.

Synopsis

erase phrase *<phrase number>* from talkfile *<talkfile number>*

Description

This command deletes the phrases identified by the phrase ID from the speech database. The phrase number many be any of the following:

- A single phrase (for example, 1)
- A set of phrases (for example, 1, 2, 5)
- A range of phrases (for example, 1–5)
- All phrases (for example, all)

After you enter the **erase** command, the system displays the following message, asking you to confirm the command before each phrase is erased:

```
Do you want to erase phrase <phrase#>? (y/n)
```

If the “all” option is used for phrases, the system prompts you only *once* to confirm the command:

```
Are you sure you want to erase ALL phrases from talkfile
<talkfile#>? (y/n)
```

If the specified phases does not exist, the system displays:

```
Phrase <phrase#> does not exist.
No action taken.
```

When the system has deleted the phrase or phrases, the system prompt is displayed.

Example

The following example erases phrase 174 from talkfile 23.

erase phrase 174 from talkfile 23

The following example erases phrases 218 through 222 and phrase 225 from talk-file 26.

erase phrase 218-222, 225 from talkfile 26

The following example erases all phrases from talkfile 29.

erase phrase all from talkfile 29

See Also

"add"
"copy"
"list"

etStub

Name

This command provides the compatibility daemon to log old **et_send**() messages into the new error logging system.

Synopsis

etStub ***[-c] [-r {rule-file}]***

etStub -s ***~{cmd}~***

Description

etStub is a daemon process which reads the IPC message queue for error messages that used to be serviced by the Error Tracker (ET) process in CONVERSANT VIS generics prior to Version 3.1. It takes each error message received and transforms it based on an ASCII rules file into a logging message for the new logging/alerting system. It is provided as a compatibility feature so that executables that worked in previous generics can be supported until the executable can be upgraded to the new logging methods.

etStub is normally started from **/etc/inittab** with an entry of the following form:

```
CVet:respawn:4:/vs/bin/vrs/etStub </dev/null >/dev/null 2>&1
```

Used in this form, **etStub** continuously reads the ET message queue and logs the messages that arrive.

The rules file that specifies the priority, destination, format, and whether the message should be flagged as obsolete for each message is normally **/vs/data/etStub.rules**. An alternate file can be specified with the *-r* flag. This would be most commonly used in conjunction with the *-c* flag, which causes **etStub** to just read the rules files, check it for syntactic correctness and immediately exit. It is recommended that any time the rules file is altered, that **etStub -c -r** ***{file}*** be used to insure that the file is correct before installing it.

etStub can be requested to reread its rules file by using the *-s* option. By executing **etStub** as a command with the *-s* option and *{cmd}* set to *rules*, a message is sent to the **etStub** causing it to discard its current rules and reread the rules file. This command can also be used to request that an alternate rules file be used or that **etStub** exit. Refer to the following examples:

```
etStub -s rules              # reread current rules file
etStub -s "rules {file}"     # read new rules from {file}. Quotes required.
etStub -s exit               # request the etStub daemon to exit & respawn
```

Notice that the quotes are required in the second example since there is a space between the command word **rules** and the file name.

Rules File

The format of the rules file is as follows:

```
#        Comments begin with the '#' character

{msg-#} {flag} {priority} {dst} ".....format of message....."
...
```

The following provides an explanation of the fields in the rules files:

{msg-#}	The number that identifies this message. It is defined by a header file found in **/att/msgipc/etmsgs**.
{flag}	This may be either 0, meaning the message is to be considered OBSOLETE and noted as such when it is logged, or -, meaning that the message will not be flagged obsolete when received and logged.
{priority}	This takes one of the following values: -, *, **, or ***C**. These correspond to NONE, MINOR, MAJOR, and CRITICAL.
{dst}	This is the destination for the message as used to be specified in the errors files. Legal values are **LOG** and **PRT**. Messages with a destination of **LOG** are sent to the MASTER_LOG files and to either the EVENT_LOG files or the ALARM_LOG files based upon the priority of the message. Those with priority NONE go to the EVENT_LOG destination in addition to the MASTER_LOG destination. All non-zero priority messages go to the ALARM_LOG destination in addition to the MASTER_LOG destination. Messages with destination of **PRT** go to the same destinations as those marked **LOG**. They also go to the **stdout** of the **etStub** process itself. If this is directed to **/dev/null** as shown in the **/etc/inittab** example, then it has no visible effect. If the **stdout** is directed to a device, such a **/dev/console**, then these messages show up at this device as well.
format	The format of the message is the same as the one that appeared in the original **/vs/data/errors** or **/gendb/data/errors** file.

Files

/vs/data/etStub.rules

Exit Values

etStub has a whole series of well defined exit values that can help in determining why it has exited. The exit values come in two forms, those used when it is running as a daemon process and those used when it is running as a command that sends messages to the daemon process. The exit values when it is running as a daemon process appear in the DEAD_PROCESS entries listed by **init** in the **/etc/utmp** and **/etc/wtmp** files. If the **etStub** daemon process is not staying up, examining the output of **who -du /etc/wtmp |grep Cvet** can provide useful information about what the problem is in addition to what is being logged to the error logging system, assuming that the **"logdaemon, reinitLog"** process is up and properly running.

Table 1-3. Exit Values of the etStub Daemon Process

1 = X_MSGRCV_FAILED	An attempt to receive a message failed. Has the message queue been removed or does it have unexpected modes?
2 = X_NO_MSGQ	The process was unable to attach a message queue. Is there more than one **etStub** daemon process?
3 = X_QKEY_BAD	Bulletin board routines are reporting that the message queue key is bad.
4 = X_MSG_TOOBIG	The message received was too big and had to be truncated. This is not an exit condition at this time.
5 = X_NO_DEVTBL	The device table shared memory does not exits. Was an attempt made to run **etStub** at run level when the VIS system was not active? Or has the shared memory been removed?
6 = X_NO_BB	The bulletin board shared memory does not exist. Was an attempt made to run **etStub** at a run level when the voice system was not active? Or has the shared memory been removed?
7 = X_LOGINIT_FAILED	An attempt to contact the **"logdaemon, reinitLog"** process failed. Is **etStub** running a level when the logdaemon is not running?

The following exit values apply when the *-s* option is being used to send commands to the **etStub** daemon process.

Table 1-4. Exit Values of the etStub Command Process

8 = X_NO_CMD	No command was provided.
9 = X_UNRECOGNIZED_-CMD	The command was not recognized.
9 = X_AMBIGUOUS_CMD	The command was ambiguous.
10 = X_WRONG_ARGS	The number of arguments provided with the command are wrong.
11 = X_TOO_LONG	The command string was too long for the message buffer and could not be transmitted to the **etStub** daemon process.
12 = X_MSGSND_FAILED	The **msgsnd ()** system call failed.

See Also

"mkETrules"

explain

Name

This command displays on-line error message explanations.

Synopsis

explain *{msgID} [...]*

explain -l *{pattern} [...]*

explain -d *{msgID} [...]*

Description

The **explain** command displays on-line error message explanations. *{msgID}* is one of the two forms of identification that comes with each message. The primary form is ***{CLASS}nnn***, where *{CLASS}* is the class of messages, such as CGEN, TSM, etc., and *nnn* is the index of the message within the class of messages. The second form, available with most messages is the mnemonic form (for example, CGEN_NOMSGQ or CGEN_MSGRCV).

If the explanation of the message fits in 24 lines and only a single explanation has been requested, it is printed without interruption. If the explanation is longer than 24 lines or more than one explanation is requested, the output is *paged* via the use of a paging program. Use the *-d* option to disable paging. The default paging program is **/bin/pg**.

If the *-l* option is used, **explain** looks up all messages whose *{msgID}* matches the pattern. For example, **explain -l A V** lists the names of explanations available that begin with either “A” or “V,” while **explain -l VROP** lists all explanation names available that begin with **VROP**. In other words, the *{pattern}* is anchored at the beginning of the *{msgID}* and assumes a match of anything after the pattern selected.

The **explain** command is also affected by certain environment variables. These environment variables are intended for advanced users only.

PAGER	The pager program used if the explanation is longer than 24 lines or more than one explanation is requested. The default is **pg**. If you do not want paging even for long explanations, using *-d* or setting **PAGER=cat** will disable paging. A one line form would be: **PAGER=cat explain {msgID}** or **explain -d {msgID}**
EXPLAINDIR	The directory in which the explanation directories are found. The default is **${PRODUCTROOT}/gendb/data/explain**.
PRODUCTROOT	This is the installation directory and defaults to ***/*** (root).
VERBOSITY	This is a debugging aid. Setting it to anything causes debugging output to be generated while **explain** performs its job.

The **"edExplain"** command allows you to add or change explanations. An explanation comes in two parts, a file containing the explanation itself, and a set of synonyms or translations that allow the **explain** command to find the file under more than one tag. To create a new explanation, you must provide both. When modifying an existing explanation, all you need to do is edit the file containing the explanation.

The explanation file itself is almost a clear text file of what you want the user to see when they ask for the explanation. There are two features of the file that are not plain clear text. All lines beginning with the “#” character are treated as internal comments and are not output. Also lines of the following form:

```
.explain {msgID}
```

have special meaning. They cause the inclusion of the explanation text specified by the *{msgID}*. This allows you to have common explanations and reference from more than one explanation.

The recommended format for an explanation procedure is:

```
 # Comment and SCCS keywords

 << {msgID} [{msgID}...] >>
{text of message}
...
```

When creating a new explanation procedure, you will be asked to edit the synonyms list and be placed in the appropriate **translateLst** file. There are instructions at the top of the file. Each non-comment line is a list of synonyms, with the right most word on the line being the name of the file in which the text is located. For example:

```
ADM001      ADM_SYSERR
ALERT003    AL_INVALID_THRESHOLD      AL_INVALID_T
```

The description for ADM001 and ADM_SYSERR are found in a file named ADM_SYSERR. The description for ALERT003 and AL_INVALID_THRESHOLD are found in a file named **AL_INVALID_T**. The second example has a truncated file name, because file names are limited to 14 characters in most UNIX systems and if you want to use source code control, then the file name must not be longer than 12 characters. The recommended way to store an explanation is under a file name related to the mnemonic *{msgID}* rather than the {*CLASS}nnn* name, since the later is meaningless. A file name of the form *{CLASS}nnn* does not provide a sophisticated user with much information about the contents of the file, while the mnemonic form does. If the mnemonic is longer than 12 characters, then you should create a shorter name related to the mnemonic that is unique within 12 characters.

There are some environment variables that affect the behavior of **"edExplain"**:

EDITOR	This is the name of your preferred text editor. The default is vi.
EXPLAINDIR	This is the directory in which the explanation directories are found. The default is **${PRODUCTROOT}/gendb/data/explain.**
PRODUCTROOT	This is the installation directory and defaults to /.
VERBOSITY	This is a debugging aid. Setting it to anything cause debugging output to be generated while **"edExplain"** performs its job.

Files

/gendb/data/explain # directory in which explanation directories are located.

/gendb/data/explain/translateLst # file containing the synonym list of {msgID}s.”

See Also

"edExplain"

fixLogFile

Name

This command upgrades existing logging files after **"lComp"** is run so that data continues to be readable by **"logCat"**.

Synopsis

fixLogFile ***[-d] [-s {save-file}] [-r] [-a] [-S] [-o {spec}] [-n {spec}]***
file1 [file2...]

Description

When classes of logging messages are expanded, contracted, inserted, or removed, **fixLogFile** can change the index assignments of messages. When this happens, messages whose indexes changed and were logged under the previous environment become unexpandable by **"logCat"**. **fixLogFile**, given information about the previous assignments and the new assignments, upgrades logged data so that it remains expandable by **"logCat"**.

Each message is examined. If the class of messages appears in the new environment and still covers the index assigned to the message, a new index is assigned based on where it appears in the new environment. If the class of messages is no longer part of the message logging environment or if a class is reduced in size so that it no longer covers the index of a message, then it is necessary to do one of three things:

- *-d* — This option deletes the message entirely from the logging file.
- *-r* — This option demaps the message. This entails expanding the message in the old environment and then creating a new logging message using the LOG_REMAP_DISCARD format so that the data is still readable in the log files, but is marked as being part of a discarded message environment. This is the default behavior.
- *-s {save_file}* — This option removes the message from the original logging file and saves it in the specified file, thus preserving the unique data for possible later retrieval.

Normally, **fixLogFile** generates a short message about each file that it converts. The *-S* flag suppresses this output.

fixLogFile requires access to the old **o.systemLog.h** and **o.textLogFmt** files and the new **systemLog.h** file to perform its job. It expects to find these files in

***$LOGROOT/*formats**. If alternate sources of these files are to be used, the *-o* and *-n* flags are used. Each of these flags takes a *{spec}* argument, which has the following form:

{dir}[,{systemLog.h}][,{textLogFmt}]]

The default values for these two specifications is:

-o ${LOGROOT}/formats,o.systemLog.h,o.textLogFmt
-n ${LOGROOT}/formats,systemLog.h,textLogFmt

The *{dir}* portion specifies an alternate directory in which the ***[o.]*systemLog.h** and ***[o.]*textLogFmt** files are to appear. If the remainder of the *{spec}* is missing, the default file names apply. If specified, the ***{systemLog.h}*** and ***{textLogFmt}*** portions specify the names of these two files as they appear in the specified *{dir}*. Any section of the specification that is skipped retains its previous or default value.

A list of one or more logging files may be specified. If they are listed, each one is assumed to be a compressed logging file and is converted. The *-a* option automatically converts all of the compressed logging files found in ***${LOGROOT}/*data**. No file names can be provided if the *-a* option is specified. When the *-a* option is used, each regular file found in ***${LOGROOT}/*data** is examined to see if it is a compressed logging file. If it is not, it is ignored. If it is, it is converted.

After the files are converted, the time stamps are reapplied so they have the same date after conversion as they did before the conversion.

Caveats

fixLogFile only takes care of changes in classes of logging message. For example, if the class PERM was added, removed, or moved, **fixLogFile** could correctly deal with the changes to the logging files. **fixLogFile** does not deal with reorganizations or changes of messages within a class. Do *not* change the order of appearance messages or the arguments to a logging message if you expect to be able to expand the data in the future or save the previous **textLogFmt** file for the expansions.

If the conversion takes place while the **"logdaemon, reinitLog"** process is running, be sure to either stop and restart **logdaemon** or reinit it using the **reinitLogD** command.

See Also

"logCat"
"logdaemon, reinitLog"

get_config

Name

This command gets the **/vs/data/conf_data** file from floppy disk.

Synopsis

/vs/bin/util/get_config

Description

The **get_config** command is used to retrieve from the CONFIGURATION DATA floppy for a particular machine the **/vs/data/conf_data** file. That file represents the configuration of the VIS machine as it was shipped from the factory or as determined by the **/vs/bin/util/configure** program after the last upgrade was performed on the machine.

The **get_config** command should be used as the first step in upgrading an existing VIS machine in the field.

Files

/vs/data/conf_data

See Also

"add_device"
"change_device"
"configure"
"remove_device"
"save_config"
"show_config"
"show_devices"

gse_add

Name

This command transfers a speech phrase from a UNIX file to the speech database in the Graphical Speech Editor (GSE) format.

Synopsis

gse_add ***<talkfile number> <phrase number> <codestyle> <input file>***

Description

The **gse_add** command transfers a speech phrase from a UNIX file to the speech database. Use this command when a phrase that does not belong to a speech pool needs to be added to a talkfile.

The *<talkfile number>* and *<phrase number>* parameters refer to the talkfile and phrase identifiers in the speech file system. The talkfile and phrase numbers of the phrases to be added must be known. Use the **"list"** command when this information is not known. The *<codestyle>* parameter can be pcm64, adpcm32, or adpcm16. The *<input file>* parameter is the file from which the phrase is to be taken.

Example

The following example adds the phrase 100 to talkfile 103 using ADPCM32 format from the file **/usr/speech/103/1000**.

```
gse_add 103 100 adpcm32 /usr/speech/103/1000
```

See Also

"gse_addpl"
"gse_copy"
"gse_copypl"

gse_addpl

Name

This command adds (restores) phrases to a specific speech pool from UNIX files in the Graphical Speech Editor (GSE) format.

Synopsis

gse_addpl ***<speech pool> <input directory> <codestyle> [<file1>...<fileN>]***

Description

The **gse_addpl** command reads the phrase list file in the speech pool (**/speech/talk/*<speech pool>*.p**l) to determine the talkfile, phrase numbers, and file names of the phrases to be added. Use the Shared Speech Pools screen for the Script Builder application to determine which speech pools are being used by the application.

The *<speech pool>* parameter is the name of the speech pool to which the speech is to be added. The *<input directory>* parameter is the name of the directory where the GSE edited files are located. The *<codestyle>* is either pcm64, adpcm16, or adpcm32. The *<file1>* through *<fileN>* parameters are the optional file names identifying the phrase names to be added. If no file names are specified on the command line, all phrases in the speech pool for which file are found in *<input directory>* are added. If file names are given, only the phrases with the particular file names that are specified in the phrase list file are added.

An unrecorded phrase is marked with a negative number in the phrase list file. If **gse_addpl** is used to added a previously unrecorded phrase, the phrase number is changed to a positive value to indicate the phrase exists. The applications using that phrase list must then be verified and installed with the specific speech pool.

Example

The following example adds phrases 1000, 1001, and 1002 to talkfile 103 using ADPCM32 format from files f1000, f1001, and f1002 in the directory /speech/talk/talk3.file

```
gse_addpl talk3 /speech/talk/talk3.files adpcm32
```

See Also

"gse_add"
"gse_copy"
"gse_copypl"

gse_copy

Name

This command extracts a speech phrase from the VIS speech file system to a UNIX file in the Graphical Speech Editor (GSE) format.

Synopsis

gse_copy *<talkfile number> <phrase number> <output file> [“<tag>”]*

Description

The **gse_copy** command extracts a speech phrase from the VIS speech file system to a UNIX file. This command may used be used when a phrase that does not belong to a speech pool needs to be edited.

The *<talkfile number>* and *<phrase number>* parameters refer to the talkfile and phrase identifiers in the speech file system. The talkfile and phrase numbers of the phrases to be added must be known. Use the **"list"** command when this information is not known. The *<output file>* parameter is the file where the speech phrase is to be placed. The *“<tag>”* parameter is an optional 50 character string that is placed into the GSE voice header of the output file. The GSE displays the tag value when the file is being edited.

NOTE:
You must keep record of which extracted files are associated with what talkfile and phrase in order to return the speech to its proper place after editing. We recommend that the *<output file>* be the same as the *<phrase number>* and the directory containing the *<output file>* be the same as the *<talkfile number>*. See the example below.

Example

The following example extracts phrase 1000 from talkfile 103 and places it in the file **/usr/speech/103/1000** for editing by the GSE.

```
gse_copy 103 1000 /usr/speech/103/1000
```

See Also

"gse_add"
"gse_addpl"
"gse_copypl"

gse_copypl

Name

The command allows multiple speech phrases to be copied from the speech file system in the Graphical Speech Editor (GSE) format.

Synopsis

gse_copypl *<speech pool> <output directory> [<file1>...<fileN>]*

Description

The **gse_copypl** command allows you to copy multiple speech phrases from the speech file system. It reads the phrase list file belonging to the speech pool (**/speech/talk/*<speech pool>*.pl**) to determine the talkfile, phrase numbers, and output file names for the phrases to be extracted. Use the Shared Speech Pools screen for the Script Builder application to determine which speech pools are being used by the application.

The *<speech pool>* parameter is the name of the speech pool from which the speech is to be retrieved. The *<output directory>* parameter is the name of the directory where the output files are to be placed. The *<file1>* through *<fileN>* parameters are the optional file names identifying the particular phrase names to be extracted. If no output file names are specified on the command line, all phrases in the speech pool are extracted. If file names are given, only phrases with those file names specified in the phrase list file are extracted.

Example

The following example extracts phrases 1000, 1001, and 1002 from talkfile 103 and places them in files **f1000**, **f1001**, and **f1002** (respectively) in the directory **/speech/talk/talk3.files**. These files are then ready to be editing using the GSE.

```
gse_copypl talk3 /speech/talk/talk3.files
```

See Also

"gse_add"
"gse_addpl"
"gse_copy"

hassign

Name

This command assigns host service to host sessions.

Synopsis

hassign *[hostsvc] <host_application>* to *<session #> [FTSCRT]*

Description

The **hassign** command assigns applications to session numbers. It is necessary to use this command to associate an application with host interactions to a given logical unit (LU). One host session corresponds to one LU.

The *FTSCRT* argument is required to assign that session for file transfer. If you are using file transfer, valid session numbers are 0–31 (that is, only sessions on the first host communication card can be used for file transfer).

The **hassign** command automatically executes the host maintenance “login” sequence on the specified session.

If the application “appl” is currently assigned to a particular session, and a new application “new_app” is assigned to that same session, the old application “app” is logged out and “new_appl” is logged in. In other words, the application currently assigned to the session is replaced by the new application you wish to assign.

If the application “my_appl” is assigned to session 0 and you wish to assign the application “your_appl” to session 0, you must:

1. **"hlogout"** the application “my_appl”
2. **"hfree"** the application “my_appl”
3. **"hlogin"** the application “your_appl”

If the application “my_appl” is already assigned to session 0 but the state is logged out and **hassign** is run on “my_appl” again, nothing happens. You need to run **"hlogin"** to log the application “my_appl” back in.

Example

In the following examples, the user is assigning the host application “my_appl” to session 0, to session 0–19, and to all sessions (up to 64 total with 2 host communications cards installed), respectively.

```
hassign my_appl to 0
hassign my_appl to 0-19
hassign my_appl to all
```

In the following example, the user is assigned in the host application “file_trans” to session 0 for file transfer.

```
hassign file_trans to 0 FTSCRT
```

See Also

"hdelete"

hdelete

Name

This command removes host service from host sessions.

Synopsis

hdelete *[hostsvc] <host_application>* from *<session number or range or all>*

Description

The **hdelete** command executes the logout sequence that is defined in the application's host session maintenance section and automatically removes the host application association from the session number. One host session corresponds to one logical unit (LU).

NOTE:
The **"hfree"** command works similarly to the **hdelete** command except that does not execute the logout sequence and should be used only when you need to release the session immediately.

Example

In the following examples, the user is unassigning the host application "my_appl" from session 0, from session 0–19, and from all sessions (up to 64 total with 2 host communications cards installed), respectively.

```
hdelete my_appl from 0
hdelete my_appl from 0-19
hdelete my_appl from all
```

See Also

"hassign"
"hfree"

hdisplay

Name

This command shows host applications which have been successfully verified and installed and the application assignments on the host sessions.

Synopsis

hdisplay ***[hostsvc]***

Description

The **hdisplay** command displays the host applications which have been verified and installed on the system. The **hdisplay** command also displays the current session assignments for each host application.

Example

The following example displays the host application currently verified and installed on the system as well as the current session assignments.

```
hdisplay
```

headFIX

Name

This command converts user application Error Tracker (ET) message rules header files used in generics prior to VIS Version 3.1 (V3.1) to the header files required for the VIS V3.1 logger/alerter environment.

Synopsis

headFIX ***[-y {year}] [-c{company}] [{class1}_et.h] [{class2}_et.h...]***

Description

“Full” conversion from the ET environment associated with VIS generics prior to V3.1 to that used by V3.1 requires that the application source be modified or converted and then recompiled. Additionally, several files used in conjunction with the ET environment must be converted to those used by the V3.1 logger/alerter.

headFIX converts the ET error rule header files to a set of header files used by the V3.1 alerter/logger. The ET rules header file reserved for applications (generics prior to V3.1) is **/att/msgipc/etmsgs/appl_et.h**; however, any properly formatted rules header file may be converted. **headFIX** converts the ET header file name *{class}_et.h* into a corresponding file ***{CLASS}*.h** which is used by the V3.1 alerter/logger. File names input into the converter must adhere to this naming convention. Following conversion, the **log*{CLASS}*.h** header file resides in your working directory. This file must be moved to the directory **/usr/spool/log/head** where i5 is used by the processes which use the alerter/logger.

If no ET header files are supplied as command arguments, a help message is displayed explaining the usage of the command and the target locations of the output. If the file argument is not found, is not named properly, or is not formatted in accordance with ET rules files, the appropriate error and help messages are displayed.

Each **log*{CLASS}*.h** file created by **headFIX** has a copyright header. By default it is for the current year and is assigned to AT&T. A alternate year can be specified using the *-y* flag and the company name can be altered using the *-c* flag.

Example

The following example converts ET headers {class1}_et.h {class2}_et.h etc.

headFIX class1_et.h class2_et.h...

See Also

"mkMsg"

hfree

Name

This command unconditionally releases host sessions from Script Builder host application assignments.

Synopsis

hfree ***[<host_application> or <session number or range or all>]***

Description

The **hfree** command releases sessions from their Script Builder application assignments. One host session corresponds to one logical unit (LU). It frees the assignment and leaves the host session on the screen it was currently.   This command can be helpful in resolving a problem with a particular screen. Normally, the **"hdelete"** command should be used to make a session available. The **hfree** command is used commonly when problems occur on sessions and troubleshoot-ing is needed.

NOTE:
The **hfree** command does not automatically log out the specified session. Use the **"hdelete"** command to log out a session and make the specified session available.

See Also

"hdelete"

hlogin

Name

This command runs the login sequence of a host script.

Synopsis

hlogin *[<host_application> or <session number or range or all>]*

Description

The **hlogin** command invokes the login procedure that is defined in the application's host session maintenance section. This command is often used in the system's cron table to automatically log in the host as soon as it is available each day.

NOTE:
The session must be in the logged out state before you may execute this command.

Example

The following example invokes the login procedures for the host application "my_appl."

hlogin my_appl

The following example invokes the login procedures for the host applications assigned to sessions 0 through 9.

hlogin 0-9

See Also

"hassign"
"hlogout"

hlogout

Name

This command runs the log out sequence of the host script.

Synopsis

hlogout ***[<host_application> or <session number or range or all>]***

Description

The **hlogout** command invokes the logout procedure that is defined in the application's host session maintenance section. This command is often used in the system's cron table in order to log off the host automatically before it goes down each night. It is a clean, convenient way to log out of the host application.

NOTE:
The session must be in the logged in state before you may execute this command.

This command should be performed once an application has been developed and hassigned to a session to test the logout procedure.

Example

The following example invokes the log out procedure for the host application "my_appl."

hlogout my_appl

The following example invokes the log out procedure for all session numbers.

hlogout all

See Also

"hassign"
"hlogin"

hnewscript

Name

This command installs a changed host script.

Synopsis

hnewscript *<host application>*

Description

The **hnewscript** command updates the system memory with the newest copy of the specified host application. This can cause the host application to be logged out and then logged back in using the newly defined host maintenance. This is required to put the updated version into effect, and Script Builder automatically prompts you when the verify and install have been composed and the host maintenance has changed.

Example

The following example updates the system memory with the most current copy of the host application “my_appl.”

```
hnewscript my_appl
```

Warnings

The **hnewscript** command may temporarily prevent access to any host sessions which have been modified while they are in the process of logging out and logging back in.

host_cfg

Name

This command translates an ASCII configuration file into a properly formatted binary configuration file.

Synopsis

host_cfg *<ASCII_config_file> <binary_config_file>*

Description

The **host_cfg** command translates the data from the specified ASCII configuration file into the format needed by the 3270 host card. This command then writes the data to the specified binary configuration file.

The *<ASCII_config_file>* argument is the name of the ASCII configuration file. This file is usually in **/usr/lib/3270/host.cfg0** or **/usr/lib/3270/host.cfg1** for card 0 and 1, respectively. These ASCII files contain configuration data for 3270 host cards 0 and 1.

The *<binary_config_file>* argument is the file name where the compiled ASCII configuration data is stored. Once the binary configuration file is created, it can be used via the **"load_bin"** command to initialize the card.

A series of keyword name = keyword value(s) entries appear in the ASCII configuration file with only one "name=value(s)" entry per line. The keyword name must start at column one on each line. An equal sign (=) should separate the keyword name from the keyword value(s). All letters in all keyword names should be upper case. A list of keyword values for a given keyword name must be separated by commas. All white spaces (spaces and tabs) are ignored. Refer to the *CONVERSANT VIS Host Interface*, 585-350-815, for additional information about configuration parameters.

Files

/usr/lib/3270/host.cfg0 **/usr/lib/3270/host.cfg1**	The ASCII configuration files
/usr/lib/3270/host.bin0 **/usr/lib/3270/host.bin1**	The binary configuration files

Example

The following example shows the contents of an ASCII configuration file for a card name “host3270a” using SNA protocol with 6 logical units (terminals), half duplex operation, cluster controller address of 193, with an XID of 0176c8c4 hex, and NRZ encoding.

```
SYNCPORT=host3270a
EXEC_TYPE=SNA
CRT24_80=2, 32, 64, 108, 187, 254
SDLC_ADDR=193
XID=0176c8c4
LINE_MODE=HALF
NRZ_CODE=NRZ
```

See Also

"load_bin"

hsend

Name

This command sends a file to the host via CVIS_FTS.

Synopsis

hsend file=*<cs_file> [dest=file_destination] [opt=option_list|n]*

Description

The **hsend** command is used to send a file to the host via cvis_fts. The arguments for the **hsend** command are:

- The *file* parameter is mandatory argument for the **hsend** command. The *<cs_file>* parameter is the full path name of the UNIX system file to be sent to the host. Refer to the *CONVERSANT VIS Host Interface*, 585-350-815, for file name guidelines for file transfer.
- The *dest* parameter is an optional argument, where *<file_destination>* is the final destination of the file at the host. If this parameter is not specified, the DESTINATION parameter value in the file **/vs/data/fts_config** is used.
- The *opt* parameter is an optional argument, where *option_list|n* is the list of option parameters or the letter *n* (for no options). Options must be separated by a space. Refer to the *CONVERSANT VIS Host Interface*, 585-350-815, for a detailed options list. If *opt=n*, the PARAM1, PARAM2, and PARAM3 values in the **/vs/data/fts_config** file are not used. If this argument is missing (the default), the PARAM1, PARAM2, and PARAM3 values in the **/vs/data/fts_config** file are used.

Return Values

Refer to the *CONVERSANT VIS Host Interface*, 585-350-815, for information on CLEO file transfer return codes.

hspy

Name

This command displays a screen currently present on a specified host session.

Synopsis

hspy ***[session number or range or all]***

Description

The **hspy** command shows what screen currently is being presented on that specified session, a range, or all. One host session corresponds to one logical unit (LU). This information helps to isolate what screens might be involved in a problem, should one occur. This tool can help to resolve problems, but should not be the only source of problem isolation.

Example

The following example displays the screen currently be presented on session 0.

```
hspy 0
```

Output

A screen of data representing what is currently present on a host session.

Warnings

This screen presents what the process that communicates with the host believes is present, but it may not be the actual screen present on that host session.

hstatus

Name

This command shows the current status of the host sessions.

Synopsis

hstatus ***[<host application> or <session_number or range or all>]***

Description

The **hstatus** command reports the current status of the host application assigned to the associated host sessions. One host session corresponds to one logical unit (LU). The possible status states are:

- Logging in — This is a temporary state indicating the session is in the process of logging in immediately after a manual **"hassign"** or **"hlogin"**.
- Logged in — This state occurs after a successful login. The session is ready to accept a transaction (the transaction base screen is reached).
- Logging out — This is a temporary state indicating the session is in the process of logging out immediately after a manual **"hlogout"**.
- Logged out — This state indicates that service is still assigned, but the session has been logged out.
- Recovering — This state occurs if the login procedure fails, the transaction ends somewhere other than the transaction base screen, or the recovery procedure ends somewhere other than the transaction base screen.
- Unassigned — This state indicates that service was never assigned to the session or service was assigned and later manually deleted.
- Not available — This state indicates the session is not available for use.
- Free — This state indicates the session was manually freed.
- Transaction — This state indicates the session is currently involved with a transaction.

This command is helpful in debugging problems with host applications and to check on the number of sessions actively involved on a call. Refer to Chapter 6, “System Monitor,” of *CONVERSANT VIS Version 4.0 Operations*, 585-350-703, for information on the Host Monitor screens.

Example

The following example displays the current status of the host applications assigned to sessions 0 through 9.

hstatus 0-9

The following example displays the current status of all session numbers.

hstatus all

iCk, iCkAdmin

Name

This **iCk** command is the daemon process which performs various integrity checks on the CONVERSANT system based on rules in a script file.

The **iCkAdmin** command is a related administration command to **iCk**.

Synopsis

iCk [-v NNN] [{envName}={value}] [{rule-file}]

iCk -c [-i | -f {file} | cmd...]

iCkCmd [-i | -f {file} | cmd...]

iCkAdmin [-c] [-a {on|off}] [s {entryType [:{ID}]}]
[-e {entryType [:{ID]}] [iCk.rules-file]

Description

iCk performs various jobs that fall into the category of "integrity" checks. It is driven by an ASCII file containing rules describing the checks desired to be performed. Its primary job is to run as a daemon process, started by **init**, and to perform the specified jobs at the intervals specified by the rules. **iCk** 's secondary job is to serve as a command interface to a human user and convey commands to the **iCk** process which is running as a daemon process.

As a daemon process, **iCk** accepts one flag, the *-v* flag, which initializes the internal verbosity flags according to the value NNN provided. This value can be in decimal, hexidecimal, or octal. None of the symbolic flag names apply in this mode. The bit meanings are as follows:

Table 1-5. Verbosity Flag Values

Bit	Name	Description
0x0001	V_RESCANBB	Log messages whenever the bulletin board is rescanned
0x0002	V_TIMINGMSG	Log messages when timing messages are sent
0x0004	V_HUNGPROCESS	Log messages when hung process checking is performed
0x0008	V_AUTOREBOOT	Log messages when autoreboot processing is performed
0x0010	V_FILEMAX	Log messages when maximum file checks are performed
0x0020	V_FILECHECK	Log messages when file ownership/modes are checked
0x0040	V_PIPECMDS	Log messages when pipe commands are received
0x0080	V_TRACE	Log messages about all major routines in **iCk**
0x0100	V_SERVICE	Log messages whenever a service is queued or performed

iCk also accepts environment variables from the command line of the form:

{variable-name}={value}

These can be used to set the following environment variables that also affect **iCk**'s behavior:

VERBOSITY	This is an alternative way to set the internal verbosity flags. The meanings of the bits are the same as for the value supplied to the *-v* flag.
SHELL	This specifies the name of the shell to be used when executing commands. The default is **/bin/sh**.

UTMP	This specifies where the “utmp” file associated with the system is located. Currently, this value is not used except for debugging purposes.
PATH	This indicates where **iCk** finds executable programs. The default is **/bin:/etc/:/usr/bin:/vs/bin:/vs/bin/util:/vs/bin/tools**.

When running as a daemon process, **iCk** accepts a file name, which is the name of the rules file from which it is supposed to operate. If not specified, the default rules file is **/vs/etc/iCk.rules**.

When **iCk** is executed with the *-c* flag or by the alternate name **iCkCmd**, is run as the command interface to the **iCk** daemon process.

-i	This option specifies that **iCk** to run in interactive mode. This causes it to generate prompts as it requests information from its standard input. Without the *-i* flag, **iCk** silently accepts input from its standard input. This might be useful if used in a shell script.
-f {file}	This value causes **iCk** to read a series of commands from the specified file or device instead of from its standard input.
{cmd}...	This field causes **iCk** to use the remaining arguments on the command line as the commands to be sent to the **iCk** daemon process.

See the “Commands” section for details about commands to which **iCk** will respond.

iCkAdmin is a command for administering the **iCk.rules** file. It has no direct communication with the **iCk** daemon process. Changes it might make to the rules file do not take effect until the **iCk** daemon process is requested to read the modified rules file.

-c	This option causes **iCkAdmin** to verbosely check out the rules file and report complaints.
a {on|off}	This option causes the rules file to be read, the **"autoreboot"** entry set the specified state, and written back out again.
-s {entryType[:{ID}]}	This option causes the rules for the specified entries to be shown.
-e {entryType[:{ID}]}	This option allows interactive editing of the specified entries. *This feature is not yet complete*.

For both the *-s* and *-e* options, the **entryType** is the name of a type of entry minus the “$” character, that is, **rescanBB**, **timingMsg**, etc. The optional *{ID}* field means the name of the process for **timingMsg** and **hungProcess** entries and the name of the file for **fileMax** and **fileCheck** entries.

Rules File

Comments begin with the “#” character and continue to the end of the line. All blank lines are ignored. Activity requests are indicated by keywords, all of which begin with the “$” character.

In the descriptions of the activities, the following definitions apply:

- *{process}* — This is the ASCII name of a process appearing in the bulletin board, that is, TSM or MTC.
- *{runlevels}* — This is specification of which run levels at which to perform the activity.The syntax is the same as used by **init**, that is, 4 = run level 4, 234 = run levels 2, 3, or 4.
- *{checkPeriod/Time}* — This indicates the activities performed repetitively will have a specification of either how often to perform the activity or at what times of the day or week to perform the activity. One of three forms is used:

-	Perform the activity once when the rules are first read and then do not perform it again.
checkPeriod	A period of time is specified as the sum of a number of different time elements: [NNd] [NNh] [NNm] [NNs]. For example, 5m means “every 5 minutes,” and 5h 30m means “every five and a half hours.” Each element is a number followed by the type of specifier, d, days, h, hours, m, minutes, s, seconds. The order is irrelevant. 5h 30m is the same as 30m 5h.
Time	If it is more important that an activity be performed at a specific time of day or week, then the “time” format should be used. It has the following form: **X {monthday} {weekday} {hour} {min} {sec}** All five elements are required for the specification to be accepted. Each element can be: * — All items in class (days of the month, hours in the day, etc.) N — The specific item.

N-M — The items between N and M inclusive.

N,M — The individual items N and M in the class.

The items within each class are:

{monthday} — 1-31
{weekday} — ASCII day of the week (sun, mon,...)
{hour} — 0-23
{min} — 0-59
{sec} — 0-59

For example: “*** * * 0 0**” means perform each hour on the hour. “**13 fri 12 0 0**” means perform the activity at noon on any Friday the 13th.

- *{cmd}* — This specific command is executed if the activity so dictates.Within the command itself, there are four meta-words that can be used to generate flexible commands. Not all four meta-words have meaning in all cases.

%f	The full file name
%d	The directory portion of the file name
%b	The base name of the file name
%p	The process identification (PID) of the process

Activities

$timingMsg *{process} {runlevels} {checkPeriod/Time}*
This activity causes a timing message to be sent to a specified process at regular intervals whenever the system is at one of a specified run levels. Currently, the TSM and the VROP processes expect to receive timing messages, once every 2 seconds.

$hungProcess *{process} {runlevels} {checkdPeriod/Time} {timeout {fill|report|exec cmd}}*
This activity causes a specific process, whose name appears in the bulletin board, to be evaluated to see if it is hung in regard to reading its messages. Processing only takes place when the system is at one of the specified run levels. *{timeout}* is the length of time the process can stay in the “working” state before being declared hung. Once a process is determined to be hung, one of three responses are possible:

kill
The process is killed by sending it first a **SIGUSR1** signal, followed by a **SIGKILL** signal if it does not voluntarily exit.

report

A message is logged to the effect that the process is hung. No other action is taken.

exec

The specified command is executed. The %p meta-word has the value of the PID of the process associated with the rule.

$autoReboot ***{off/on} {u-reboots} {ubPeriod} {runlevels} {setPeriod}***

This activity controls the feature which automatically sets the UNIX kernel auto-reboot flag. If the entry is marked "off," then the auto-reboot flag is not automatically turned on. It can still be manually set with an **iCk** command. If the entry is marked "on," then the automatic setting is enabled. The remaining parameters control when the flag is set. The algorithm that controls the setting of the flag is as follows:

1. The number of unanticipated reboots of the kernel is determined by examining the **/etc/wtmp** file (the history file of **init** actions) for "change of run level" entries and "boot time" entries. Any entry falling within the *{ubPeriod}* of time prior to the most recent system boot time are considered. If a "boot time" entry is preceded by a "change of run level" to levels 0, 5, or 6, the boot is considered anticipated, since someone deliberately entered the command to reboot the system. If a "boot time" entry is NOT preceded by such a "change of run level" entry, then the reboot is considered unanticipated. This includes power failures, reset button pushes, and panics of the UNIX kernel.
2. If the number of unanticipated reboots is LESS than *{u-reboots}*, the auto-reboot flag is set *{setPeriod}* amount of time AFTER the system comes up to one of the run levels specified by *{runlevels}*.
3. If the number of unanticipated reboots is GREATER THAN OR EQUAL to *{u-reboots}*, setting of the auto-reboot flag is inhibited and is not set until the system has been up at one of the run levels specified by *{runlevels}* for a *{ubPeriod}* of time.

For example: typing **$autoReboot on 5 60m 4 5m**,which is the standard default setting specifies that if LESS THAN 5unanticipated reboots have occurred in the past 60 minutes, the auto-reboot flag is set in the UNIX kernel 5 minutes after reaching run level 4. If 5 or more unanticipated reboots have occurred in the past 60 minutes, then the auto-reboot flag is not set until 60 minutes after reaching run level 4.

$fileMax *{file} {maxSize} {checkPeriod/Time} reduce {minSize}*
$fileMax *{file} {maxSize} {checkPeriod/Time} remove*
$fileMax *{file} {maxSize} {checkPeriod/Time} exec {cmd}*

This activity checks one or more files to insure that they have not grown too large. *{file}* is the name of a file or a pattern specifying a set of files. *{maxSize}* is the maximum size in bytes that a file to grow to before it triggers a response from **iCk**. A check on the size of the file or files is made as specified by *{checkPeriod/Time}*. One of three responses to a file becoming too large an occur:

reduce
: The offending file is reduced in size by saving the last *{minSize}* bytes of the file and discarding the rest.

remove
: The offending file is removed entirely.

exec
: The command specified is executed. In this case the meta-words **%f**, **%d**, and **%b** are defined as the various parts of the file name and can be used in the command.

$fileCheck *{file} {runlevels} {checkPeriod/Time} {type} {owner} {groups {modemask} {modes} [cmd]*

This activity can be used to insure that a specific file or files exist and have the proper ownership and modes. *{file}* specifies the file or a pattern which selects a set of patterns. *{runlevels}* specify at which run levels the checks are made. {*checkPeriod/Time}* specifies the frequency of checks. *{type}* specifies the type of file. It can take one of seven values:

-	The type does not matter.
f	The file is a “regular” file.
d	The file is a directory.
p	The file is a named pipe.
c	The file is a character special device.
b	The file is a block special device.
l	On SVR5.4 systems, the file is a symbolic link.

{owner} specifies who owns the file. If this value is - then who owns the files is not of interest. *{group}* specifies which group owns the file. If this value is - then which group owns the files is not of interest. *{modeMask}* specifies which bits of the mode are of interest while *{modes}* is the state of the bits desired. For example, if both *{modeMask}* and *{modes}* were 0444, then the check would be to insure that the file was readable by anyone, but whether it was writable or executable is not of interest. If on the other hand *{modeMask}* was 0777, while *{modes}* was 0444, then the check would be to insure that the file was only readable and must not be writable or executable by anyone. If a file fails to pass a $fileCheck test, it is always reported. If the optional *[cmd]* is specified, then this command is executed. The meta-words %f, %d, and %b are set to the various parts of the file name for use in the command.

$EOF

This special mark indicates the end of the rules. Anything beyond this mark in the rules file is ignored.

Example Rules

$fileMax /etc/wtmp 360000 ~* * * 0 0~ reduce 36000

If the file **/etc/wtmp** exceeds 360,000 bytes, reduce it to 36,000 bytes. Check the size of the file on the hour. (The structures in this file are 36 bytes in length and it must be an integral number of structures, hence the chosen sizes.)

$fileCheck /etc/passwd - - f root - 0777 0444

Check only once. The **/etc/passwd** file should be owned by **root** and be read-only to everyone.

$fileCheck /etc/shadow - - f root - 0777 0400

Check only once. The **/etc/shadow** file should be owned by **root** and be read-only only to **root**.

$fileMax /tmp/*.lst 10000 - remove

Remove all the files in **/tmp** ending with an extension of *.lst* if they are bigger than 10,000 bytes. Do this only once.

$fileMax /tmp/*.hist 0 - exex ~/bin/mv %f %d/o.%b~

For any non-zero length files in **/tmp** with an extension of *.hist*, save them as **/tmp/o.*.hist**

Commands

In command mode **iCk** responds to the following commands. Each command sends a message to the **iCk** daemon process except for the first command. All commands can be abbreviated to the shortest unique string, hence **au** is sufficient to identify the **"autoreboot"** function and **ac** the **activate** function. For most commands one letter is sufficient.

x " | " exit " | " ^D
This command exits from the interactive command mode. This does not affect the **iCk** daemon process.

bootCnts [period]
This command computes the UNIX reboot information from the **/etc/wtmp** file. If *period* is supplied, this length of time is used. If it is not supplied, then the window period of time for the **$autoReboot** rule is used. This command generates three numbers, the total number of reboots in the specified period of time *prior* and including the current boot of the system, the number that were anticipated (or deliberate) and the number of unanticipated reboots. This request does not communicate with the **iCk** daemon process.

"autoreboot" {set|clear}
This command forces the kernel auto-reboot flag into the specified state.

readRules [rule-file]
This command rereads the rules file. If a new file name is provided, then it is read instead of the previous file. Before using this command, the new rules should be checked with the **iCkAdmin** command to insure syntactic correctness.

wakeup
This command makes the **iCk** daemon wakeup immediately and check its state.

rescanBB
This command makes the **iCk** daemon wakeup and reexamine the bulletin board for new instances of known process types.

quit
This command causes the **iCk** daemon to exit gracefully. (Since **iCk** is normally run from the **/etc/inittab** file, **init** immediately respawns the daemon.) In interactive mode, the command requires confirmation.

verbosity ***{value}***

This command sets the **iCk** daemon's verbosity flags to the specified values. In this case the symbolic names are accepted as well as octal, decimal, or hexidecimal values. Combined values can be produced by separating values with the '|' character.

The following three commands **activate**, **inhibit**, and **print** require an activities specification. Such a specification is defined from the following list of objects. More than one object can be combined with the '|' character:

rescanBB

This object is the **$rescanBB** activity.

timingMsg

This object is all the **$timingMsg** activities.

hungProcess

This object is all the **$hungProcess** activities.

"autoreboot"

This object is the **$autoReboot** activity.

fileMax

This object is all the **$fileMax** activities.

fileCheck

This object is all the **$fileCheck** activities.

miscellaneous

This object applies to the **print** command only. It causes a report of whether the autoreboot flag has been automatically set or not, the state of the UNIX kernel autoreboot flag, the current run level, the number of rules read, and the number activities currently in force to be logged.

all | ALL

This object specifies all activities.

NNN

This object, where NNN are digits, specifies an explicit activity by its index in the array of all activities.

activate ***{spec}***

This object, in conjunction with the V_TRACE flag, causes the activities specified by *{spec}* to be logged whenever they execute.

inhibit ***{spec}***

This object, in conjunction with the V_TRACE flag, causes the activities specified by *{spec}* to not be logged whenever they execute.

print *{spec}*

This object logs the status of the activities specified by *{spec}*. The status information logged as a result of the **print** command varies based on the activity. The common information printed is the activity index, which may be used in future *{spec}* ‘s, the rule index, which should correspond to the position of the rule in the rules file, and the type of the activity. In addition, there is the *a_clockID*, which is non-zero if an alarm is running for the current activity and the **a_nextAlarm**, which indicates at what time the next alarm is set to expire. At the end of the entry is the **a_flags**, 0, meaning no flags are set, AF_SUPPRESS_TIMING, meaning that timing is deliberately suppressed for the time being, AF_CHECK_NEW_RUNLEVEL, meaning that when the run levels change, this activity is checked to see if it should reactivate, and AF_DEBUG_OFF, which is set for any activity that has been inhibited by the **inhibit** command. There is also the **a_state**, which indicates the current state of the activity. Its values are:

AS_INACTIVE — This value is currently not being processed.

AS_TIMER_RUNNING — There is currently an alarm outstanding for this activity.

AS_SERVICE_QUEUED — An alarm has expired for this activity, but has not yet been processed.

AS_IN_PROGRESS — An activity is currently being processed.

All remaining information is activity specific. By activity the information logged is:

$timingMsg

The name of the process, the bulletin board slot, and instance.

$hungProcess

The name of the process, the PID, the bulletin board state, work count, time, flag, slot, and instance. The flag can have values of HP_STUCK, meaning that it does not seem to be reading its message queue, HP_SIGUSR1, meaning it has been sent a SIGUSR1 signal to request it to die, and HP_SIGKILL, meaning that it has been killed with the uncatchable SIGKILL signal.

$autoReboot

The computed unanticipated reboot count at the time the system was last rebooted plus the length of the period over which the computation is made.

$fileMax

The name of the file.

$fileCheck

The name of the file.

core

This command is available for debugging purposes. It causes **iCk** to produce a core file in **/tmp/iCk.core** via a core dump operation is a spawned child process. In other words, **iCk** itself does not stop, but you do get a reliable core of **iCk** for debugging evaluation.

Default File

iCk responds to default parameters placed in **/vs/etc/default/iCk**. Initially there are two values, which set specific internal parameters:

RUNLEVELTIMEOUT

This parameter specifies how long to wait after changing run levels before accepting the value from **/etc/utmp** without confirmation from **iCkCmd**. The default is 3 minutes.

RECHECKTIMEOUT

This parameter specifies how long to wait after changing run levels before rechecking for new processes in the bulletin board. The default is 30 seconds.

Also any environment variables desired can be set in the default file.

Files

/vs/etc/iCk.rules	# the default rules file
/tmp/iCkPipe	# the named pipe used to speak to iCk
/vs/etc/default/iCk	# default parameters

Caveats

iCk is a daemon process running as “root.” Since the rules support the concept of executing an arbitrary command, the **/vs** and the **/vs/etc** directories need to be protected against tampering and the **iCk.rules** file should only be writable by authorized users.

See Also

"logCat"

install_appl

Name

This command installs an application.

NOTE:
This command is valid only if the Enhanced File Transfer package is installed.

Synopsis

install_appl -n *[o] <application name>*

Description

The **install_appl** command is used to install a verified application received from the host, bundled using the **"backup_appl"** command. It requires the name of the application. The *[o]* option overwrites the existing application.

NOTE:
You must use the **"restore_appl"** command before using the **install_appl** command.

Return Values

If the **install_appl** command is successful, a 0 value is returned. If any value other than 0 is returned, the **install_appl** command failed. The following are the possible reasons for failure for the **install_appl** command:

- The hard disk is low in space.
- The command syntax is incorrect.
- The application already exists.
- The application has not been verified.
- Unrecorded phrases exist.
- An inconsistency is found in the transaction.

Example

The following example installs the application “bank_balance” received from the host.

```
install_appl -n bank_balance
```

See Also

"backup_appl"
"remove_appl"
"restore_appl"

install_remote

Name

This command sets up and activates the remote login port and modem.

Synopsis

install_remote /dev/tty/ ***[00|h0[1-8]] [port speed]***

Description

The **install_remote** command puts the modem in the auto-answer mode and performs other associated tasks. These associated tasks include checking the validity of the command line arguments, permitting the remote login port to remain up, saving the new port name and speed, updating the reboot time command that puts the modem in the auto-answer mode, and initializing the dialup password feature.

NOTE:
Once you have completed the **install_remote** command, do not alter the port configuration via the FACE menus or use the **"removepkg"** command to remove the serial port card software. If you must change the port configuration, use the **"remove_remote"** command first.

The */dev/tty* option is the remote login port name. If the asynchronous card is not installed, only the “/dev/tty00” port is available. If the asynchronous card is installed, one of the card’s eight ports (that is, “/dev/ttyh01”, “/dev/ttyh02”, etc) is available.

The *port speed* option can be either 1200 bps, 2400 bps, or 9600 bps. The default is 1200 bps.

Files

/etc/rlogin
This file contains the remote login port name and speed data

/vs/bin/initmodem
This file puts the modem into the auto-answer mode.

/etc/dpasswd
This file sets up the dialup password feature for the remote login port.

Example

The following example sets up and activates the remote login port “/dev/tty1” at 1200 bps.

```
install_remote /dev/tty1 1200
```

The following example sets up and activates the remote login port “/dev/tty4” at 9600 bps.

```
install_remote /dev/tty4 9600
```

See Also

"remove_remote"

install_sw

Name

This command installs a software package.

NOTE:
This command is valid only if the Enhanced File Transfer package is installed.

Synopsis

install_sw ***[-p <path>] [-n <cpio file name>]***

Description

The **install_sw** command is used to install a software package received from the host. The package is a file in cpio format. If the path and the cpio file name are not specified, the default path (**/tmp/stag**) and the default file name (**h_install**) is used.

Use the following command to view the contents of a floppy to determine if it is an installable package and that is does not use full path names, like **/etc/profile**:

cpio -iBcvtl /dev/rdsk/f0

The following files are needed to perform the UNIX **"installpkg"** command: **Size, Name, Files, Install, "remove"**, and ***<package name>***.

The following steps creates a sample “bundled” cpio file called fts from an installable UNIX package:

1. Insert the first floppy of the package to be bundled in the floppy disk drive.
2. Enter **mkdir mydir**
3. Enter **cd mydir**
4. Enter **cpio -iBacvdl /dev/rdsk/f0**
5. Enter **find . -print | cpio -ocd > fts**
6. Transfer this binary file to the target system using file transfer or enhanced file transfer.
7. Enter **install_sw** on the target system.

Return Values

If the **install_sw** command is successful, a 0 value is returned. If any value other than 0 is returned, the **install_sw** command failed. The following are the possible reasons for failure for the **install_sw** command:

- The hard disk is low in space.
- The user is not root or a super user.
- The command cannot read “path>/<name>.”
- The package already exists.
- The command syntax is incorrect.
- The package is missing the necessary installation programs.

Example

The following example installs the “sbpkg” cpio file which contains the CONVERSANT Script Builder Version 4.0 package.

```
install_sw -n sbpkg
```

See Also

"remove_sw"

installpkg

Name

This command installs a software package.

Synopsis

```
installpkg
```

Description

The **installpkg** command loads a software package from floppy disk to the hard disk. You must be logged in as **root** to execute the **installpkg** command.

The **installpkg** command performs the same functions as installing software through the AT&T FACE menus. It is recommended that **installpkg** be used when loading software into a new system. Once a system is running, use the AT&T FACE menus to load any additional optional software packages.

Example

The following example invokes the program to load a software package from floppy disk to the hard disk.

```
installpkg
```

into_et

Name

This command sends error messages to the error tracker (ET).

NOTE:
This command is obsolete in the logger/alerter environment (post Version 3.1). Refer to the **"logit"** command for additional information on sending error messages.

Synopsis

into_et ***[-fhlpstv] [-n no_messages] [-r error no_times]***

Description

The **into_et** command sends the error messages to ET and is used primarily to verify that the specified errors are being recognized by ET. Once errors are sent, display the Event Log Report in the **"cvis_menu"** system to see if ET logged them appropriately.

The error messages can be specified as the numeric message id or as the mnemonic (the default). After processing its options (except for the -r option), **into_et** goes into interactive mode where error messages can be entered one per line using the following format:

mnemonic channel card e_arg[0] e_arg[1] e_arg[2] e_arg[3] e_strarg

or

message id

The input fields above are separated by tabs and/or spaces and are optional except for the mnemonic or message id. The mnemonic should be exactly as defined in the **/att/msgipc/etmsgs** directory's files or **into_et** rejects it. ET applies the e_arg and e_strarg fields to the rule associated with the error to form the text description; they are the same arguments that are passed to **et_send**.

To quit **into_et**, you may:

1. Enter a lower case letter for the mnemonic.
2. Enter a number less than 100 for the message id.
3. Press DELETE.

The **into_et** command recognizes the following options:

- *-a* — This option sends all message ids known to **into_et**. **Into_et** sends 20 at a time, then sleeps for 5 seconds before continuing with the next 20.
- *-f* — This option gives a fuller prompt message instead of the default terse message while in interactive mode.
- *-h* — This option displays a help message.
- *-l* — This option accepts message ids as input instead of mnemonics (the default).
- *-n no_messages* — This option sends at most the specified maximum number of messages. **Into_et** exits after the number of messages sent equals no_messages.
- *-p* — This option displays all known messages ids or mnemonics. It sends all message ids known to **into_et**. **Into_et** sends 20 at a time, then sleeps for 5 seconds before continuing with the next 20.
- *-r* — This option sends the specified error message the specified number of times. The error argument is either a mnemonic or message id depending on whether option -1 is specified. **into_et** exits after the last message is sent and thus it skips interactive mode.
- *-s* — This option accepts an unknown mnemonic and send message id 17 that is unknown currently to ET.
- *-t* — This option sends a message in file into.t in the current directory. Each line in into.t specifies an error message to be sent, having the same fields as when specifying error messages interactively (see above). However, note that **into_et** interprets the first field as a mnemonic only; the first field should be a message id.
- *-v* — This option turns on verbose mode. All arguments sent to ET are displayed.

ET expects every error message received to have a source of the error. Error messages sent **into_et** have ET as the source of the error. In effect, **into_et** pretends to be ET.

Upon successful completion, **into_et** displays the total number of error messages sent and an exit status of zero is returned. Otherwise, **into_et** prints an error message on stdout and returns a non-zero exit status.

NOTE:
After the initial prompt, **into_et** no longer prompts for a error message which makes it difficult to tell if the message has been sent and whether to proceed to enter the next error message. If the cursor rests on the beginning of the next line, you can enter the next error message.

NOTE:
The **into_et** messages are subject to flood control, meaning that multiple messages of the same type are not shown in the error log unless flood control is turned off. See the discussion of **etset** for more information.

Examples

In the following examples '"cr>" stands for carriage return. Assume that mnemonics (message id) DIP_READ_ERROR (1440) and DIP_HOST_ACCESS (1450) are known while BAD_ARGS and BAD_NAME are unknown to **into_et**.

Example 1: The following example sends DIP_READ_ERROR to ET twice.

```
into_et -f <cr>
DIP_READ_ERROR <cr>
DIP_READ_ERROR <cr>
q <cr>
```

Example 2: The following example is an alternate way to send the DIP_READ_ERROR to ET twice with verbose mode on.

```
into_et -v -r DIP_READ_ERROR 2 <cr>
```

Example 3: The following example is a third way to send DIP_READ_ERROR to ET twice>

```
into_et -l <cr>
1440 <cr>
1440 <cr>
1 <cr>
```

Example 4: The following example sends DIP_HOST_ACCESS, DIP_READ_ERROR, and BAD_ARGS to ET.

```
into_et -s <cr>
DIP_HOST_ACCESS <cr>
DIP_READ_ERROR <cr>
BAD_ARGS <cr>
q <cr>
```

See Also

et_send

lComp

Name

This command combines a series of message files and produces a file of compressed format files and an expansion format file.

Synopsis

lComp ***[-s name] [-c name] [-t name] [-d name] [-m name]***
<file1> [file2...]

Description

lComp compiles logging format files. The input files are in the form:

```
XXX...NNN... message.....%fff[<<SQL spec>>]....
%fff[<<SQL spec>>]....%fff[<<SQL spec>>]...
```

In other words, the input files contain standard C format statements, with optional SQL field definitions included. Long lines may be broken up with backslash, newline sequences. Such lines are concatenated, discarding the backslash and newline characters, by **lComp** and treated as one long line during compilation.

lComp produces five files, a header file, a compressed format file, an expansion format file, a data dictionary file, and a data dictionary mapping file. The default names are: **systemLog.h, cmpLogFmt, textLogFmt, dataDictLog**, and **ddMapLog**.

-s name	Changes the **systemLog.h** file to ***name***
-c name	Changes the **cmpLogFmt** file to ***name***
-t name	Changes the **textLogFmt** file to ***name***
-d name	Changes the **dataDictLog** file to ***name***
-m name	Changes the **ddMapLog** file to ***name***

The **systemLog.h** file contains a series of defines of the form:

#define _*{FILE}*_START NN

where *{FILE}* is the all uppercase form of the input file name. This header file allows applications to refer to errors of a specific class relative to the beginning of the class of errors and so avoid having to edit code as the various classes of errors codes grow or shrink.

The **cmpLogFmt** file contains the compressed formats, which the log subroutine uses to produce compressed logging messages.

The **textLogFmt** file contains two sections. The first section is a series of offsets to each expansion format and its length. The second section contains the expansion formats, which **expandLog** uses to convert a compressed logging file into a human readable statement.

The **dataDictLog** file contains SQL names for the variable fields in each message. They are of the form:

abs_index <FS>fld-name,type[,length[,precision]]<FS>...<GS>

The *abs_index* is the index number of the message within the universe of all messages compiled by **lComp**. If the optional SQL specification does not appear after the format, **lComp** generates one of the form:

CLASSNNN_M,type[,len[,precision]]

based on the format. *CLASS* is the uppercase name of the file the message came from, *NNN* is the index of the message within the file, and *M* is the field within the message, starting at 1.

The **ddMapLog** file contains structures describing where to find each data dictionary entry for each message. It also contains an array with the class names.

See Also

"logCat"

list

Name

This command lists the directory entries for specific phrases.

Synopsis

list phrase *<phrase number>* in talkfile *<talkfile number>*

Description

The **list** phrase command displays the phrases stored in the specified talkfile. The valid arguments for the **list** phrase command are:

- *<phrase number>* — This argument specifies the number (or range) of phrase(s) to be listed.
- *<talkfile number>* — This argument specifies the number (or range) of talk-file(s) containing phrase(s) to be listed.

The listed entries are sorted by talkfile number and phrase. The information printed for each phrase consists of talkfile number, phrase number, phrase size in bytes, phrase size in blocks, the phrase length in seconds, and the speech coding type.

Example

The following example displays phrase 174 as stored in talkfile 25.

lis phrase 174 in talkfile 25

The following example displays phrase 12 as stored in talkfile 1.

lis phrase 12 in talkfile 1

See Also

"add"
"audit"
"copy"
"erase"

load_bin

Name

This command initializes the 3270 host card.

Synopsis

load_bin ***<binary_config_file> <card_number>***

Description

The **load_bin** command loads the executable program and binary configuration files into the specified 3270 host card for the VIS. **load_bin** reads in the specified binary configuration file to obtain the configuration data and executable program indication. It also resets the appropriate 3270 host card, downloads the appropriate executable program to the card, downloads the binary configuration data to the card, sends the device data to the device driver, and starts the card.

The *<binary_config_file>* argument is the filename of the binary configuration file. The binary configuration file is usually stored in **/usr/lib/3270/host.bin0** or **/usr/lib/3270/host.bin1** for card 0 and 1, respectively. Binary configuration files are produces using the **"host_cfg"** command.

The **load_bin** command should be used only when the voice system is not running and not taking calls as this command resets the sessions on the card. For dialup links, the link is dropped after executing **load_bin**.

The *<card_number>* argument is a single digit value. Values are 0 or 1. A 0 represents 3270 host card number 0 and 1 represents 3270 host card number 1.

Example

The following example requests card 0 be loaded using the specified configuration file:

```
load_bin /usr/lib/3270/host.bin0 0
```

Files

/usr/lib/3270/host.bin0	The binary configuration file for card 0
/usr/lib/3270/host.bin1	The binary configuration file for card 1

See Also

"host_cfg"

logCat

Name

This command reads the compressed logging files and outputs human readable messages.

Synopsis

logCat ***[-{t|b} lines] [-a locant] [-z locant] [-v] [-c] [-m] [-r root]***
[-s locant] [-q locant] [-w width] [-p continuation-prefix]
[-d data -l log-prefix | file] [-f format] [-V]

Description

logCat reads in a file of compressed logging messages generated by *log* and expands them to human readable form.

The default behavior with no arguments is to list all log files of the type specified first in the Config file, that is **logCat -d${LOGROOT}/data -l{primary-log-prefix}.**

- *-t lines* — This option tails the last “lines” of file.
- *-b lines* — This option shows beginning “lines” of file.
- *-v* — This option specifies the verbose mode (that is, report the file names of the files examined).
- *-c* — This option continuously displays the last lines of file. If the **logDaemon** switches to a new file, follow it.
- *-m* — This option is the meticulous time check. Normally, the log file name and the creation date are used to determine the date of the file. If the creation dates have been messed up, the *-m* flag causes the time stamp of the first message in each log file to be used instead of the name and modification date. This is slower, but more reliable.
- *-r root* — This option specifies an alternate root directory for **textLogFmt** file. The default is **/usr/spool/log**. Also, the **data** directory containing the compressed logging data files is expected to be in the root directory if not overridden by the *-d* flag or the LOGDATA environment variable.
- *-a locant* — This option specifies the place to start printing.
- *-z locant* — This option specifies the place to stop printing.
- *-s locant* — This option searches for specific patterns or times.

- *-q locant* — This option searches for specific patterns or times. This is the same as *-s* if the locant is a time locant. If the locant is a search pattern, the search is applied to the raw compressed log data instead of the expanded log data. This means that the pattern can only include variable portions of the logged messages. It is much faster than the *-s* option when properly applied.

  A locant is one of two things, either a date/time stamp OR a search pattern.

  Dates can be any of the standard human readable forms: mmm dd, yyyy, mm/dd/yy, mm-dd-yy, etc. The time is hh:mm:ss. It is also possible to specify the separate elements as: sec=nn min=nn hour=nn mday=mm mon=nn or mon=mmm year=nn[nn] wday=n or wday=ddd yday=nnn. Portions left out default to this date, 0 hours, 0 minutes, and 0 seconds, that is, giving only the time of day indicates today's date. If the form "item=xxx" form is used, all elements not specified default to '*', hence "wday=Sun" means all messages on any Sunday. Do not mix standard format with the "item=xxx" format. The results are not predictable.

  Spaces should be enclosed in quotes, for example, -a"7/14/87 05:08:30". Search patterns are enclosed in '/' characters, with an optional repetition count following, for example, -z/GEN006/2 means the second message containing GEN006. The repetition count has no meaning with the *-s* or *-q* locants, but does for the *-a* and *-z* locants.

  The search capability supports the following meta-search constructs:

^	Beginning of message
$	End of message
*	Any number of unspecified characters
?	A single unspecified character
[xxx]	Any character in the list "xxx"
[!xxx]	Any character not in the list "xxx"
\{chr}	Normal C backslash conventions, \n \t \b \f \r \NNN \\ \/ \[

- *-w width* — If lines are to be wrapped, this is the width at which the wrapping should take place. 0 means no wrapping and is the default. The width can also be supplied via the environment variable LOGCOLUMN

- *-p continuation-prefix* — This is the string to be appended to each continuation line. The default is no continuation prefix. The continuation prefix can also be provided via the environment variable LOGCONTPREFIX data

- *-d data* — This option is the name of the directory to find the log files in. The data directory can be provided in the environment variable: LOGDATA. The default is ***${LOGROOT}*/data**. The *-d* argument takes precedence over the environment variable.

- *-l log-prefix* — Prefix of the log files to examine. The default is the first log file in the **Config** file. The log-prefix can also be provided via the environment variable LOGFILEPREFIX file
- *file* — Explicit file to be displayed. If "-", use standard input. The use of a file name overrides the *-d* and *-l* options.
- *-f format* — Format specification for printing messages. The default is %P %T %N %S:%L\n%M

  The format specifier uses the following notations:

%P(...)	Priority level format: %d or %s
%T(...)	Time level format: all options supported by "date" command
%N	Name of process specified by the **loginit** call of the process
%S	Source file name
%L	Line number
%M	Message text
%%	The % character
\{chr}	Standard C backslash conventions
...	All other characters are printed as is. The format can be provided via the environment variable LOGFORMAT

- -V — This option makes the control characters visible. They are printed as \X if they have a special C notation, otherwise as <NNN>, where NNN is the octal value.

Environment Variables

Environment variables are checked whenever the related command argument is missing from the command line. If both the command argument and the environment variable are missing, the specified default is used.

LOGROOT
This variable is the directory in which the **textLogFmt** is found, containing the expansion formats. Also, the directory in which the **data** directory is found if LOGDATA is not specified.

LOGDATA
This variable is the directory in which the log data files are to be found. The default is ***$(LOGROOT*** **/data**.

LOGFORMAT
This variable is the format in which to print the log messages. The default is **%P %T %N %S:%L n%M**

LOGCOLUMN
This variable is the column at which to wrap long expansions. The default is 0, meaning do not wrap long messages.

LOGCONTPREFIX
This variable is the string to be prepended to continuation lines when long lines are being wrapped. The default is no prefix.

LOGFILEPREFIX
This variable is the logfile prefix to be examined in no *-l* argument is specified. If neither a *-l* argument is specified nor LOGFILEPREFIX set, then the first log destination in the Config file of the type ‘L’ is used.

See Also

"lComp"

logdaemon, reinitLog

Name

These commands are the logging daemon and a control program, respectively.

Synopsis

logdaemon [VAR=VALUE...]

reinitLog

Description

logdaemon is the general purpose logging daemon. It creates and reads ***$LOGROOT/*logpipe**. Any process wanting to log messages sends them via **"logit"** or the **logEvent ()**, **logperror ()**, or **vlogEvent ()** calls to **logdaemon**. **logdaemon** in turn sends the messages to the destinations for which they are intended.

logdaemon uses the environment variables LOGROOT and TZ. LOGROOT defaults to **/usr/spool/log** if it is not reset to something else. TZ defaults to “EST5EDT” if it is not reset to something else. These variables can be set on the command line. Any argument of the form X=Y is placed in the environment of the **logdaemon**. If **logdaemon** is being run from **/etc/inittab**, then an appropriate entry for a California site might be:

```
log:234:respawn:/pas/prsm/bin/logdaemon TZ=PST5PDT 2>/dev/console
```

logdaemon generates no I/O activity on the standard input or output. In fact, it closes both of these file descriptors as its first activity. It generates output on the standard error output if it is directed to the console or some similar location.

If ***$LOGROOT/*Config** is changed or if **"fixLogFile"** is run on a compressed logging file which is current and **logdaemon** is running, **reinitLog** sends a message which causes the **logdaemon** to reread the configuration file and reopen the various logging files.

Files

$LOGROOT/Config	The configuration file which defines destinations.
$LOGROOT/data	The directory in which logging files are created.
$LOGROOT/logpipe	The FIFO which **logdaemon** reads.

See Also

"fixLogFile"
"logCat"
"logit"

logDstPri

Name

This command creates the shared memory containing the dynamic destinations and priorities of logging messages using the **logMsg()** interface.

Synopsis

logDstPri ***[-H {dir}] [-c] [-v] [-d] [-x {cnt}] [rules]***

Description

logDstPri reads an ASCII rules file, described in **msgDst**, and then sets up a shared memory segment using the information in the rules file so that any process in the system using the **logMsg (), vlogMsg ()**, or **logSysError ()** library calls can determine the appropriate priority and logging destinations for each message they send.

By default the rules files is expected to appear in ***${LOGROOT}*****/msgDst.rules**, where *${LOGROOT}* is **/usr/spool/log**. By default the header files used to translate ASCII names of message indices into numbers are expected to appear in the directory ***${LOGROOT}*****/head**. An alternate directory for the header files can be specified via the *-H* option on the command line. An alternate rules file can be specified as a file name on the command line.

After changing the rules file, it is recommended that the rules be checked before they are put into service. The *-c* flag causes **logDstPri** to read the rules file and report any rules that are misformatted or not understood. The return value from **logDstPri** is the number of errors detected.

To see the error complaints and install the rules all at once, specify the *-v* flag. This causes the verbose complaints to be generated. The *-c* flag implies the *-v* flag.

When **logDstPri** is resetting the values in shared memory, as opposed to creating the shared memory for the first time, it can be requested to delete the old shared memory and create a new segment by specifying the *-d* flag. Do not use the *-d* flag on a running system because any process that is already using the old shared memory continues to use it even after it is “deleted.” This means that two different rules files might be in force at the same time. It may be necessary to specify the *-d* flag if a large number of new messages have been added to the rules file. Currently, **logDstPri** creates the shared memory 200 entries larger than the highest logging message index found it its rule file. This means that as long and the new rules file does not go beyond 200 entries higher than the current highest entry, everything is okay. The number of extra entries can be altered by specifying the -*x* option.

Files

${LOGROOT}	Default is **/usr/spool/log**
${LOGROOT}/msgDst.rules	The message priority and destination file
${LOGROOT}/head/*.h	Header files used by the logging system

Shared Memory Segment

The shared memory segment is keyed off the inode of the rules file and the define symbol *LDP_KEY*, defined in **log/head/logDstPri.h.** The library routine **ftok({file},LDP_KEY)** is used to generate the shared memory key.

See Also

"logCat"
logMsg
"logDstPri"
msgDst

logFmt

Name

This command displays and changes the parameters used to display messages and explanation texts, specifically the messages mnemonics and screen width.

Synopsis

logFmt ***[global] {display|interactive|{opt}={value}}***

Description

Each logging message has a class name and a mnemonic name associated with it. A class name, for example ICK001, is the combination of the name of the class, for example, ICK, and the index of the message within the class, for example, 001. The mnemonic name is a short composite string of characters which identifies the type of logging message. The mnemonic name for ICK001 is ICK_BAD_CMD. By default the mnemonic names of messages are not displayed when **"display messages"** is used to examine the logging files. If you want the mnemonic message names to appear, then **logFmt** allows you to alter the system so that they either appear for everyone by default or appear for you specifically. You can also adjust the width of the screen display. By default the screen width is set to 75 characters. If you have a wider screen, you may wish to specify that more of the screen be used to display messages.

global	This modifier causes the action specified to operate on the “global” (system wide) parameters that control the behavior of **display message**. You must be **root** if you want to change the global parameters. You can examine the global parameters without being **root**.
display	This verb causes **logFmt** to display the current parameters. If *global* is specified, then the system-wide parameters are displayed, otherwise your personal parameters are displayed.
defaults	Specifying *defaults* without the *global* option causes your personal preferences about mnemonics and screen width to be removed. You then get the system-wide settings. Specifying *defaults* with the *global* option causes the system-wide settings to be reset so that mnemonics are off and the default screen width is 75 characters.

interactive	This option interactively prompts for the parameters controlled by **logFmt**. Pressing ENTER in response to any query causes the current value to be retained. The current value appears within **[]** s.
mnenonics=enable	This option causes mnemonics to be displayed when logging messages are examined with **"display messages"**.
mnemonics=disable	This option causes mnemonics to not be displayed when logging messages are examined with **"display messages"**.
width=NN	This option causes the screen width to be set to NN, where NN is between 40 and 199 columns. The default setting is 75. Do not attempt to set the screen width to a value wider than your screen can actually handle or the display will be unpleasant when using **"display messages".**

When mnemonics are enabled, they also show up when **"explain"** is used to examine the description of a message. Whether mnemonics are enabled or not, the mnemonic name can always be used to select an explanation using **"explain"**.

Files

/vs/data/logFmtParms	# Global parameters file
${HOME}/.logFmtParms	# User's parameter file
/usr/spool/log/textLogFmt	# Current default expansion format file
/usr/spool/log/textLogFmt.Mne	# Expansion file with mnemonics
/usr/spool/log/textLogFmt.NoM	# Expansion file without mnemonics

Examples

The following example enables the mnemonics. This affects only you and overrides the system-wide setting.

```
logFmt mnemonics=enable
```

The following example sets the system wide default so that mnemonics are not displayed. Any user wishing to see mnemonics has to personally enable mnemonics. You need to be **root** to execute this command.

```
logFmt global mnemonics=disable
```

The following example displays the system wide settings for mnemonics and screen width.

```
logFmt global display
```

The following example sets your personal screen width to 130 characters when displaying messages using **"display messages"**.

```
logFmt width=130
```

logit

Name

This command represents a program used to log arbitrary messages to the logging files.

Synopsis

logit ***[-p {priority}] [-d {destination}] [message...]***

Description

logit sends an arbitrary message to the logging files and logs it under GEN002. By default, the priority is E_NONE, which is 0. Alternate priorities for the logged message can be selected with the *-p* option. Legal values for *{priority}* are 0–4 or none, manual, minor, major, or critical. The message is identified as the current user.

By default, messages generated by **logit** are sent to the MASTER_LOG which is associated with the first destination defined in the ***$LOGROOT/*****Config** file (normally, the flat log files with the prefix **log**). Messages are also generated in the expanded form on SYSDBG so that you can see what has been logged. Alternate destinations may be specified with the *-d* flag. The destination specification may be numeric (for example, 0x11) or as a series of symbolic names taken from the list MASTER_LOG, SYSDBG, SCREEN, SYSCONS, RFSTATS, and ALERTERMSG separated by "+." The destination can also be a mix of both styles. Regardless of the specification of the destination, MASTER_LOG is always added to the list.

Example

Enter the following at the system prompt:

logit -p3 -d SYSDBG+0x20 Sample major alarm

System response:

```
** Tue Oct 22 20:43:09 1992 LOGIT logit.c:136
GEN002 abc: Sample major alarm
```

The output appears in the MASTER_LOG by default and sent to the alerter pipe, as 0x20 is equivalent to destination ALERTERMSGS.

See Also

"logTest"

logTest

Name

This command is a program which can generate an arbitrary list of logging messages to be sent to the logdaemon.

Synopsis

logTest ***[“-s dir”] [-v] [-x] [“file1 file2...”]***

Description

logTest reads a script of logging messages to be sent to the **logdaemon** process. The format of the script is:

{interval} {dst} {priority} {process} {index} [arg arg...]

Following is a description of each of the values used in the **logTest** script.

- *{interval}* — This value is the number of seconds to wait before sending this message. If this is 0, then the message is sent immediately.
- *{dst}* — This value is the destination mask. Each destination is specified in the Config file for the logging system. Normally, this value will be 0x01, meaning logging the message is the primary log files only. The standard VIS Config file looks like:

```
#          @(#)Config              4.1.1.1 20:48:10 12/19/93

L C 100000 log 10                  # dst bit 0x0001
D E 0 stderr                       # dst bit 0x0002
S E 0 SCREEN                       # dst bit 0x0004
T E 0 /dev/console                 # dst bit 0x0008
N C 0 $LOGROOT/alertPipe           # dst bit 0x0010
L C 100000 alarm 10                # dst bit 0x0020
L C 100000 event 10                # dst bit 0x0040
N C 0 $LOGROOT/sccsCtlPipe         # dst bit 0x0080
N C 0 $LOGROOT/mxmtrPipe           # dst bit 0x0100

$EOF
```

  Therefore, to send something to the “alarm” log and the “sccsCtlPipe” destinations, *dst* should be set to 0xa0, the combination of 0x20 and 0x80. 0x01, the master log, is added automatically to all destinations.

NOTE:
Realize that messages sent with **logTest** are independent of the logging priority and destination mechanism that makes use of **/usr/spool/log/msgDst.rules**. With **logTest** you can send a message to any destination with any priority as anybody. It is truly a test tool.

- *{priority}* — This is a value from 0 to 4, specifying the priority of the message. 0 is E_NONE, 1 is E_MANUAL, 2 is E_MINOR, 3 is E_MAJOR, and 4 is E_CRITICAL. See the **log.h** file for more details.
- *{process}* — This value is the name of the process to be forged into the logging message. Normally, each process attaches an identifying string to each message it logs. *{process}* allows **logTest** to act like any process.
- *{index}* — This value specifies the log message index associated with the message of interest. It can be either the absolute index of the message, which might be obtained by examining a compressed log file with a text editor (it is the first field of each message), or the relative name of the message index in the form **log*TYPE(index***) where *TYPE* is the module associated with the message, (for example, GEN, LOG, SYS, PERM, etc), and *index* is the relative location of the message within the module. **logGEN(20)** means the twentieth message in the GEN category, or it can be the symbolic name of the message, (for example, **LG_NUMARGS**).
- *[arg]* — This value is the argument required by the message format. The number of arg elements depends upon the specific logging message chosen. There should be one argument for each "%" argument specifier in the **log/formats/{TYPE}msg** file that describes the logging message. If the argument include spaces, it should enclosed in double quotes. Given the following formats:

SYS001 EPERM	**File protection (modify) error %s**
SYS002 ENOENT	**No such file or directory. %s**
SYS003 ESRCH	**No such process for kill or ptrace. %s**
PERM12 PERM_SHADOW	**%s %s %s by %s**
PERM13 PERM_PROFILE	**%s's .profile %s by %s**

  the following lines would be appropriate as input to **logTest**:

  ```
   3  0x01  2  doghouse  logSYS(2)      fidget.c
  10  0x01  0  editUser  logPERM(12)  linda  "added to"  /etc/shadow  root
   0  0x01  0  editUser  logPERM(13)  linda  modified  root
  ```

  The first line is logged in 3 seconds, from the process "doghouse," saying that it could not open or create the file **fidget.** The second line comes 10 seconds later, indicating that "editUser" added "linda" to the shadow password file and that **root** made the change. The third line comes immediately and says that linda's .profile was modified by **root**.

NOTE:
No checking is done on the number of arguments required. It is the responsibility of the user to supply the proper number. If the wrong number of arguments is provided, **"logCat"** complains when trying to explain the message, saying that there is an expansion failure.

The *-v* flag causes **logTest** to send copies of the messages it is going to log to the standard error output. The *-x* option prevents **logTest** from sending the messages to **logdaemon**. *-x* automatically implies and enables *-v*, since if it does not send the message to **logdaemon**, the only other thing it can do is list it. *-x* is used to test run a script without actually logging all the messages.

To be able to use the symbolic names for the message indicies, it is necessary that **logTest** be able to read a copy of the appropriate **systemLog.h** file created by **"lComp"**. If **systemLog.h** does not exist is the current directory, the *-s* switch specifies the directory in which **logTest** can find **systemLog.h**. If **logTest** cannot find **systemLog.h**, only absolute log indicies are accepted in the input.

See Also

"logit"

lsboard

Name

This command displays the configuration information for all 3270 cards on the system.

Synopsis

lsboard

Description

The **lsboard** command displays the configuration information for all 3270 cards on the system. You must be logged in as **root** (superuser) before using the **lsboard** command.

If the **"addboard"** complains of an I/O address conflict, the problem could be an I/O address problem or a RAM address problem. Check the I/O and RAM address on the host card or cards using the **lsboard** command to be sure that no other device is using those addresses.

Example

The following is an example of the system output to the **lsboard** command:

```
Device      I/O  IRQ  Address  Size  Board-Type     KeyName
-------     ---  ---  -------  ----  ----------     -------
/dev/host0  2A0   3   D0000    16K   DataTalker-XL  pcxlbrd0
```

See Also

"addboard"
"rmboard"

mkAlerter

Name

This command reads an alerter description and generates C or C++ code which implements the description.

Synopsis

mkAlerter ***[-M] [-o {executable}] [-p {templ-path}] [-t [-f]] [-q] [-v] [-I] [X=Y...] [{alerterfile}.A...]***

Description

mkAlerter is a program that reads an alerter description and translates it, with the help of code template files, into compilable C or C++ code. It also produces a make file for compiling the code. Alerter description files always have a ".A" extension. By default **mkAlerter** produces a single source file, with an extension of ".c". It also produces a header file (extension ".h") and a make file (extension ".mk"). If the make file already exists, **mkAlerter** does not overwrite the existing file. This allows you to modify the make file as desired without fear of it being destroyed the next time **mkAlerter** is used, but does take advantage of the knowledge contained in the make file template used by **mkAlerter** when it does create a make file. The source file and the header are *always* overwritten each time **mkAlerter** is run. No modifications should ever be made to these intermediate source files, since the changes are lost the next time **mkAlerter** is run. If the *-M* flag is specified at execution time, **mkAlerter** splits the source file produced into two pieces, one containing *main ()* and the other containing everything else. The source file containing *main ()* ends in "*Main.c*" with truncation as is necessary. Once produced, this file, like the make file, is be overwritten. If you wish to produce your own initialization, you can use the *-M* option and then make your changes to the "**Main.c*" file.

Normally, the make file specifies that the executable to be produced by this alerter description is the same as the name of the alerter description minus the "*.A"* extension. The *-o* option allows you to specify an alternate executable name. This is used when the make file is generated.

The code template files are normally expected to exist either in the current directory or in **/usr/lib/alerter**. If the templates are not found in either of these places, **mkAlerter** uses its own internal copies, but also reports the fact. If the templates exist elsewhere, an alternate path can be specified with the *-p* option. Each directory which should be searched is separated by ':' characters, the same as a normal UNIX PATH description.

To get the initial template files, the user can specify the *-t* option. This causes **mkAlerter** to create each of the required template files using its internal copies. At this point each site may, if desired, alter these templates to produce alerter code appropriate for its needs. By itself the *-t* flag does overwrite existing template files. The *-f* flag causes the new templates to overwrite existing ones.

The current list of template files and their contents follows:

AlertInc.t	Description of include files.
AlertCopyR.t	Copyright notice.
AlertHead.t	Template of the header file.
AlertMain.t	Description of main () function.
Alerter.t	Primary template describing the alerter program.
AlertTest.t	Description of the code to respond to timeouts for alerting.
AlertMsg.t	Template describing a subroutine to process messages for a particular logging destination.
AlertDir.t	Template describing the subroutine to handle logging messages sent directly to the alerter process.
AlertMk.t	Template for the makefile.
AlertObj.t	Template for each *.o file in the makefile.

The *-q* option is not currently implemented. It is meant to check the templates for completeness. The *-v* flag increases the verbosity of **mkAlerter** while it performs some of its activities.

Normally **mkAlerter** produces *#line* directives, which are used by the C compiler to report where errors are detected during compilation. While these are good during the compiling phase, they mislead most debuggers and make debugging difficult. The *-l* option suppresses the *#line* directives and is recommended when the debugging phase includes the use of a process debugger, such as **sdb** or **pi**.

It is also possible to specify variable assignments that appears in the make file via the X=Y syntax. Of particular interest is CC=CC, which also causes **mkAlerter** to generate C++ code rather than C code.

See Also

readAlerterDesc

mkerr

Name

This command converts user application error tracker (ET) message rules used in generics prior to VIS Version 3.1 (V3.1) to a file suitable for use with the VIS V3.1 “transparent” logger/alerter environment.

NOTE:
This command is obsolete for the Version 3.1 new logger/alerter environment.

Synopsis

mkerr ***[-o] [-n] [error_file]***

mkerr ***[var]***

mkerr ***[-h]***

Description

The “transparent” ET environment allows application code compiled for VIS generics prior to V3.1 to run without modifications, conversion, or recompilation. It does suffer from the fact that old style messages do not appear in the same format as the new messages in terms of numbering or layout.

The transparent ET environment requires that a rules file be built from the original ET errors files used to drive the earlier ET supported systems. The process **"etStub"** uses the file **/vs/data/etStub.rules** for this transparent conversion. **mkerr** can be used to create the file **/vs/data/etStub.rules** from the user application ET rules file **gendb/data/errors** found in previous generics.

mkerr creates **etStub.rules** from the *{error_file}* provided as an argument. The command output file is automatically installed in the directory **/vs/data**. If no *{error_file}* is supplied, a help message is displayed explaining the usage of the command and the target of the output. If the *{error_file}* file does not exist or is not formatted in accordances with the expected ET rules file, appropriate error and help messages are displayed.

The *-o* option flags each occurrence of a message as “obsolete.” The *-n* option causes the rules to be generated without the “obsolete” flag set; hence the messages generated by these rules are not flagged as OBSOLETE. The default setting is *-o*.

The *-var* option automatically converts the **/gendb/data/errors** file and installs the result in the directory **/vs/data**. This option insures compatibility with the intent of this same command name and option in previous VIS generics.

Example

The following example converts the ET rule file errors to the Version 3.1 logging environment.

```
mkerr errors
```

mkETrules

Name

This command provides the compatibility daemon to log old **et_send()** messages into the new error logging system.

Synopsis

mkETrules ***[-o|-n] [{ET-errors-file...}]***

Description

mkETrules is a shell problem which produces a valid **etStub.rules** file from one or more errors files of the type that used to control the ET process. It takes as input one or more of the old style **errors** files and generates a file called **etStub.rules** in the current directory. By default all entries are created with the *O* (obsolete) flag set. The *-o* flag specifies that the obsolete flag be set explicitly. The *-n* flag specifies that the rules created not be marked with the “obsolete” flag.

See Also

"etStub"

mkheader

Name

This command allocates user memory for script variables.

Synopsis

mkheader *<application name>*

Description

The mkheader program creates an address in user memory for each script variable. This information is stored in an ***application-name*** **def.h** header file and is used in naming both the output file and the allocation program. The joint usage of the same header file enables the script to interact with the transaction state machine (TSM). The *-e* option specifies exact string matches.

The **mkheader** program prompts an operator to enter three types of information at the system console. The information may be entered interactively or batched together in a single file. Interactive entries are ended by entering CTRL D. The system prompts for:

- Variable names
- Header file names in order of dependency
- Structure names with header file locations

When **mkheader** is entered with an argument (limited to 7 characters) for *application-name*, an ***application-name*def.h** header file is created for the output information. The **mkheader** program then prompts for three types of information which it uses in producing the output file.

1. It prompts the operator for the name of one of the variables - char, int, or short. Char is the only variable which requires a length (default = 1).

   It then allocates space for the variables at the beginning of the allowable user memory and places this information in the newly created header file.

2. Mkheader prompts the operator to enter header files which are needed in order to make the files covered in the third section compile. They should be named in the order of dependence. For example, if information in the header file **b.h** is needed by the header file **a.h**, header file **b.h** must be entered first and then header file **a.h**.

Full pathnames must be given. The file **mesg.h** and the structure mbhdr are common to all scripts and are entered automatically.

The header files can be stored in a batch file. The batch file could contain the following header files.

#include "/att/msgipc/dbcom.h"
#include "/att/include/shmemtab.h"
#include "/att/msgipc/tsm stop.h"
#include "/att/msgipc/cdata.h"

3. The last prompt is used for allocating the space for each structure.The operator is prompted to enter each header file name and its structure names. For each header file, the operator enters the word all (if all structures are needed) or specific structure names.

   Mkheader recursively allocates memory and produces ***application-name*** **def.h** defines for structure members which are themselves structures (except for struct mbhdr).

As a shortcut, the input for the three prompts may be stored in another file (data file) and read in each time. For example:

mkheader ***application_name*** **< data file**

Once the header files have been entered, mkheader writes a program called ***application_name*_aloc.c** to allocate the rest of user memory. The resulting source code is automatically compiled, using **mkheader.a** library functions, and then executed. This adds the remaining structure definitions to the ***application_name*def.h** header file. TSM does not allow a script to use more than 50,000 bytes of user memory. Scripts that exceed this limit are not run when data beyond the limit are accessed.

Files

/vs/bin/vs/mkheader
/vs/bin/vrs/mkheader.a

Examples

The following are examples of the prompts and the output for the mkheader program. This example shows a user who needs some space for 20 characters, 2 integers, and a short variable. The user also needs to have space declared for a structure called dowj, which is used by the script. The header file is found in **/att/msgipc/tsmdipappl.h.**

In the example, the structure size of SZDOWJ is 16, which is automatically supplied by mkheader.

console input: mkheader <application_name>

FIRST PROMPT: Type in the variables you need space for according to the following format:

type name [length]

Example 1: int yn
Example 2: char dg 20

(End input with CTRL-D)

Variable?: char dg 20
Variable?: int yn
Variable?: short cid
Variable?: int iom

Variable?: (CTRL-D)

SECOND PROMPT: Please enter any dependency files that the header files in the next section will need in order to compile. Use full path names. (End input with CTRL-D)
File name? /u/factory/file.h
File name? (CTRL-D)

THIRD PROMPT: Enter the header file name and structure names needed to create the def.h file. Use full path names. (End input with CTRL-D)
Header file?: /att/msgipc/tsmdipappl.h
Structures or all?: dowj
Header file?: (CTRL-D)

Compiling: application-name aloc.c
Running: application-name aloc
Output is called: application-name def.h

This is the final ***application_name*****def.h** file produced by this example.

```
/*****PRE-ALLOCATION OF USER SPACE *****/

#define DG:0
#define YN:20
#define CID:24
#define IOM:26

/***** DOWJ STRUCTURE *****/

#define DOWJ:30
#define RCODE:30
#define TIMEDATE:31
#define CATNUM:42
#define MKTSTAT:43
#define DOWHOUR:44
#define SZDOWJ:16
```

In this second example, the command line includes a data file from which the system gets the information usually entered by the users in response to system prompts.

The data file, called "data" in this example, contains the following information:

```
char name 20
int answer
short reply
^D
/att/include/shmemtab.h
^D
/att/msgipc/cdata.h
Day_pntr cdata
^D
```

The following appears on the screen:

```
Conversant% mkheader test6 < data
Type in the variables you need space for according to
the following format:
     type name [length]

     Example 1: int yn
     Example 2: char dg 20

(End input with CTRL-D)

Variable?:
Variable?:
Variable?:
Variable?:

Please enter any dependency files that the header files
in the next section will need in order to compile.
Use full path names.
(End input with CTRL-D)

File name?: File name?:

Enter the header file names and structure names needed
to create the def.h file. Use full path names.
(End input with CTRL-D)
Header file?: List of structures or all?:Header file?:

Compiling /usr/has/another/test6_aloc.c

Running /usr/has/another/test6_aloc

Output is called /usr/has/another/test6def.h

I am now checking for any duplicate defines that will
cause problems

The following is the contents of the test6def.h file:

/******* PRE-ALLOCATION OF USER SPACE *******/

#define NAME:0
#define ANSWER:20
#define REPLY:24
```

```
/******* DAY_PNTR STRUCTURE *******/

#define DAY_PNTR              26
#define FILE_FIRST            26
#define REC_FIRST             28
#define FILE_LAST             30
#define REC_LAST              32
#define SZDAY_PNTR             8

/******* CDATA STRUCTURE *******/

#define CDATA                 34
#define SCRIPT                34
#define CHAN                  50
#define EQUIP                 52
#define STARTTIME             54
#define STOPTIME              58
#define EV0                   62
#define EV1                   66
#define EV2                   70
#define EV3                   74
-:
-:
-:
#define EV96                 446
#define EV97                 450
#define EV98                 454
#define EV99                 458
#define SZCDATA              428
```

NOTE:
Make sure that all variable names are unique without respect to case as lower case letters are changed to upper case for the final output.

mkimage

Name

This command performs a system backup.

Synopsis

mkimage *[-t | -f] [all|files|speech]*

Description

The **mkimage** command performs a system backup by copying the UNIX files in the **root** and **usr** file systems and the speech data to the backup media specified.

NOTE:
Specifying floppy disks as the media when using this command is very time consuming. We recommended that you back up to cartridge tape.

The following options are used with the **mkimage** command:

-t	This option backs up the data to cartridge tape.
-f	This option backs up the data to floppy disk.
all	This option backs up the data in the **root** and **usr** file systems and the speech data.
files	This option backs up the data in the **root** and **usr** file systems.
speech	This option backs up the speech data.

Example

The following example backs up the **root** and **usr** file system to cartridge tape:

```
mkimage -t files
```

mkMsg

Name

This command converts user application error tracker (ET) message rules used in generics prior to VIS Version 3.1 (V3.1) to the files required for the VIS V3.1 full logger/alerter environment.

Synopsis

mkMsg ***[-y {year}] [-c {company}] {error_file1} [{error_file2}...]***

mkMsg -h

Description

Full conversion from the ET environment associated with VIS generics prior to Version 3.1 (V3.1) requires that the application source be modified or converted and then recompiled. Additionally, several files used in conjunction with the ET environment must be converted to those used by the logger/alerter.

mkMsg converts the ET error rules files to a set of files used by the V3.1 alerter/ logger. The ET rules file reserved for applications (generics prior to V3.1) is **/gendb/data/errors**; however, any properly formatted rules file may be converted. Prior to reformatting via **mkMsg**, you must separate your ET rules files that correspond to message classes, each class being defined by an ET message header file. The naming convention for this header file was ***{class}*****_et.h**. The header file corresponding to **/gendb/data/errors** was /**att/msgipc/etmsgs/appl_et.h**. **mkMsg** converts the ET file named *{CLASS}* into a corresponding file ***{CLASS}*****msg** which is used by the V3.1 alerter/logger. To maintain a consistent naming convention, we recommended that the **/gendb/data/errors** file be copied to **appl** prior to conversion. Following conversion, the ***{CLASS}*****msg** should be copied into **/usr/spool/log/formats** where it can be used by the alerter/logger. The **/usr/spool/log/formats/formats.mk** should also be modified to include the new ***{CLASS}*****msg** file.

If no ET message files are supplied as command arguments, a help message is displayed explaining the usage of the command and the target locations of the output. If the file argument is not found or is not formatted in accordance with ET rules files, appropriate error and help messages are displayed.

Each ***{CLASS}*****msg** file created by **mkMsg** has a copyright header. By default it is for the current year and is assigned to “AT&T.” An alternate year can be specified using the *-y* flag and the company name can be altered by using the *-c* flag.

Example

The following example convert the ET rules files file1 file2....

```
mkMsg file1 file2...
```

See Also

"headFIX"

msgadm

Name

This program facilitates the administration of system messages in the VIS application software.

Synopsis

msgadm ***[-e] <[-f <[<input_file>|-]>|<command>]>***

Description

The **msgadm** provides an interface to the CONVERSANT VIS logger/alerter administrative files. Commands to **msgadm** may be specified individually on the command line using **msgadm <*command*>** or may be specified as input from a file or standard input using the *-f* flag and a file name argument as in **msgadm -f *<filename>*** or **msgadm -f -** for file input and standard input respectively. The *-e* flag forces **msgadm** to write $EOT after completing each operation resulting in command output to standard out.

Each command may require one or more of the following variable arguments:

- *<message_ID>* — This argument specifies a member of the set of system message IDs which include all those whose message class is indexed through the **systemLog.h** file and whose mnemonics appear in a configured **log*XXX*.h** file.
- *<priority>* — This argument specifies a priority tag as defined with the $priority operator in the **/vs/data/msgDst.rules** file. Use **msgadm** priorities to see a list of priority tags configured with the system.
- *<time>* — This argument is a non-zero positive integer indicating time in seconds if suffixed by “s,” minutes if suffixed by “m” or hours if suffixed by “h.”
- *<dst>* — This argument specifies the set of destination tags defined in the **/usr/spool/log/msgDst.rules** file through the $destination operator. To see the list, execute **msgadm** destinations. Note that only the latest destination specified in **/usr/spool/log/msgDst.rules** is used.
- *<threshold>* — This argument specifies a non-zero positive integer indicating a threshold value.

The following are examples of how the **msgadm** command may be used:

- **msgadm set <*message_ID*> priority <*priority*>** — This example sets the priority of *<message_ID>* to *<priority>* if *<message_ID>* is already in the **msgDst.rules** file. If *<message_ID>* does not exist, an entry is created with the indicated priority and the default destination(s).
- **msgadm <[add|delete]> <*message_ID*> destination <*dst*>** — This example, if add is specified and <*message_ID*> exists in the **msgDst.rules** file, adds a new destination entry to the file. If the entry does not exists a new entry is created with the default destination plus the specified destination. The priority is set to the default priority. If delete is specified, <dst> is removed from the destination set for <*message_ID*>. The log or MASTER_LOG destination cannot be removed from a message.
- **msgadm set <*message_ID*> window <*time*>** — This example sets the threshold window time of threshold *<message_ID>* to *<time>*. If no threshold structure has yet been created for *<message_ID>*, one is created with a threshold of 100 and threshold message set to THR001.
- **msgadm <[add|delete]> <*message_ID*> threshold <*threshold*> message <*thresh_message_ID*>** — This example adds or deletes a <*threshold*>/<*thres_message_ID*> pair from the **thresh.rules** file. If add is specified, and no entry for <*message_ID*> exists, an entry is created with a threshold window of 1 hour. If delete is specified and the <*threshold*>/ <*thres_message_ID*> pair is the last pair specified, the entire threshold structure is removed.
- **msgadm display <[<*message_ID*>|*all*]>** — This example lists all administrative parameters associated with <*message_ID*> or all system messages if *all* is specified.
- **msgadm priorities** — This example outputs the default list of message priorities. **msgadm** destinations outputs the default list of message destinations.
- **msgadm thresholds** — This example outputs the set of thresholding messages.
- **msgadm sync** — This example makes all changes made through previous calls to **msgadm** take affect in the live logger/alerter system. If sync is not used, a system reboot is required to make changes take affect.
- **msgadm -f <[<*input_file*>|-]>** — This example forces **msgadm** to read from <*input_file*> or standard in if “-” is specified. The expected input is **msgadm** command line arguments as defined above, one complete set of command line arguments is expected per line. Errors in the input result in no changes to the logger/alerter configuration files regardless of where the error occurred in the input.

Examples

The following example sets the priority of system message VROP003 to critical. Note that *C has been quoted to protect it from the shell.

msgadm set VROP003 priority '*C'

The following example adds a threshold of 10 with a threshold message of THR001 to the thresholding structure for the VROP003 message. It is assumed that THR001 is a valid message Id.

msgadm add VROP003 threshold 10 message THR001

The following example displays the message administration parameters associated with message VROP003.

msgadm display VROP003

System response:

```
       Message Id:              VROP003 (VROP_NOSPBUF)
       Message Priority:        *
       Message Destinations:    log|alarm

       Threshold Period:        1m
       Message Thresholds:
         Threshold              Threshold Message Id
         ---------              --------------------
                20               THR003 (THRESH_MAJOR)

        Message Text:
VROP003  -- -- --- (VROP_NOSPBUF) No resources available on
        SP for speech %s.  Reason:  No SP window buffers.
```

The following example shows the use of the file input mechanism. This example sets message VROP003 to priority “-” (none), changes its destination from the alarm to the event and removes its thresholding structures (if any exist). It then sets TSM002’s priority to “*C” (critical), assuming “*C” is defined in the **msgDst.rules** file, and makes the changes take effect in the current environment through the sync directive.

```
message -f - <<!
set VROP003 priority -
delete VROP003 destination alarm
add VROP003 destination event
delete VROP003 threshold 10 message THR003
delete VROP003 threshold 100 message THR004
set TSM002 priority ‘*C’
sync
!
```

See Also

"display messages"
"explain"

newscript

Name

This command updates the changes to all currently assigned scripts.

Synopsis

newscript

Description

The **newscript** command notifies the TSM and CDH processes that an existing script in the **/vs/trans** directory has been changed. After **newscript** is run, TSM reloads all scripts from disk when they are run next instead of using any copies it has in memory.

Example

The following example notifies the TSM and CDH processes that an existing application in the directory **/vs/trans** has changed.

```
newscript
```

Files

/vs/bin/util/newscript

remove

Name

This command places a unit in the manual-out-of-service state.

Synopsis

remove *<unit> <number> [-i] [-n]*

rem *<unit> <number> [-i] [-n]*

Description

This command is used to remove a unit from service when its temporary state is idle. It changes the permanent state of the unit to manual-out-of-service (MANOOS). It does not remove a unit that has a temporary state of busy. If a unit must be interrupted immediately or appears to be stuck busy, use the **rem *<unit> <number>* immed** command.

The parameters are for the **remove** command are:

- *<unit>* — This parameter identifies the unit. The choices are “channel” or “card.”
- *<number>* —This parameter specifies the channel or card number, a range of channel or card numbers in the form m n, or the word “all” for all the channel or card numbers. Card numbers are in the form ***card#[.port#]*** where *port#* is a port of card *card#*. If *port#* is not given, all ports of the card specified are removed. If no card number or channel number is given, a syntax message is displayed.
- *-n* — This option disables prompting from the system whether to wait until a conflict has been resolved (see the *-i* option below) or to terminate the request to remove.
- *-i* — This option is used to enable secondary command registration. If T1 diagnostics are being run, this option allows the “removing” of another card. If -i is used and another maintenance command is being run **("remove", "detach", "attach", "restore", diagnose**), the request to **remove** card is blocked and a message is printed to the screen. If *-i* is not used and any maintenance command is being run, the request to **remove** card is blocked and a message is printed to the screen.

If the command is permitted to run, a check is made to see if the command is in conflict with another. A command is in conflict if the card or card associated with it:

1. Is the T1 card being diagnosed
2. Will cause a change in the existing TDM bus master assignment
3. Has an interdependency with the T1 card being diagnosed (for example, PRI)

If one of the above conflicts exist and *-n* is not used, the user is asked whether to wait until the conflict is resolved or to terminate the request. If T1 diagnostics are executing on-line tests and a conflict is detected, the **remove card** command is blocked. If T1 diagnostics are executing off-line tests and a conflict is detected, the user is asked whether to wait until the conflict is resolved or to terminate the request to remove.

To delete out of the command, press DELETE. If this does not stop the command, you may need to press CTRL and backslash simultaneously. If, while running remove, you wish to abort the command, a message similar to the following may appear:

```
At the user’s request, administration of the following
cmd(s) has been interrupted.

CARD NUMBERS: <card numbers>

To assure proper operation of the identified card(s),
run diagnostics at the earliest opportunity.
```

It is recommended when **remove** is aborted, diagnostics be run on all cards being administered to ensure they are returned to a fully functional state.

Example

The following example removes card 0 from service.

rem card 0

The following example removes channels 0 through 2 and channel 4 from service.

rem channel 0-2,4

The following example removes all cards from service.

rem card all

See Also

"attach"
"detach"
"restore"

remove_appl

Name

This command removes an application.

NOTE:
This command is valid only if the Enhanced File Transfer package is installed.

Synopsis

remove_appl ***[-d|s|t|f] -n <application name>***

Description

The **remove_appl** command is used to remove an application. Only one of the following options is allowed at one time.

-d	This option removes the database tables.
-s	This option removes the speech.
-t	This option removes the transaction.
-f	This option removes the installed files.

If no option is specified, the whole application is removed.

Return Values

If the **remove_appl** command is successful, a 0 value is returned.If any value other than 0 is returned, the **remove_appl** command failed. The following are the possible reasons for failure for the **remove_appl** command:

- The hard disk is low in space.
- The command syntax is incorrect.
- The voice system is not running.
- The command to remove the database tables failed.
- The command to remove the installed files failed.
- The command to remove the speech failed.
- The command to remove the transaction failed.

Example

The following example removes the application “bank_balance.”

```
remove_appl -n bank_balance
```

See Also

"backup_appl"
"install_appl"
"restore_appl"

remove_device

Name

This command removes a device from the **/vs/data/device_data** file.

Synopsis

/vs/bin/util/remove_device

Description

The **remove_device** command is an interactive menu driven program that allows a user to remove a device entry from the **/vs/data/device_data** file. Only those devices which have been added may be deleted. The standard set of default devices with which the file is initially populated may not be removed.

Removing a device from the **device_data** file ends support of that device by the **/vs/bin/util/configure** program. The removed device is no longer available for possible configuration by the **"configure"** program.

Files

/vs/data/device_data

See Also

"add_device"
"change_device"
"configure"
"get_config"
"save_config"
"show_config"
"show_devices"

remove_remote

Name

This command removes the remote login modem from the auto-answer mode.

Synopsis

remove_remote

Description

The **remove_remote** command takes the modem out of the auto-answer mode and performs other associated tasks. These tasks include disallowing the remote login port to remain up, removing reboot-time command, deleting the remote login port name and speed, and removing the dialup password feature.

Files

/etc/rlogin	This file contains the remote login port name and speed data.
/etc/resetmodem	This file removes the modem from the auto-answer mode.

See Also

"install_remote"

remove_sw

Name

This command removes an installed package.

NOTE:
This command is valid only if the Enhanced File Transfer package is installed.

Synopsis

remove_sw ***<package name>***

Description

The **remove_sw** command is used to remove any of the installed software package.

The *<package name>* argument is the name that appears when a displaypky is executed and should be enclosed in double quotes (“ “).

Return Value

If the **remove_sw** command is successful, a 0 value is returned. If any value other than 0 is returned, the **remove_sw** command failed. The following are the possible reasons for failure of the **remove_sw** command:

- The hard disk is low in space.
- You are not logged in as root or super user.
- The package name is not specified.
- The package does not exist.
- The command can not find the removal script for the package.

Example

The following example removes the software package “CONVERSANT Script Builder Version 4.0.”

```
remove_sw “CONVERSANT Script Builder Version 4.0”
```

See Also

"install_sw"

removepkg

Name

This command removes a software package.

Synopsis

removepkg

Description

The **removepkg** command removes a software package from the hard disk. You must have superuser privileges to execute the **removepkg** command.

When you enter the **removepkg** the VIS displays a list of the software packages installed on that machine and prompts you to specify the package you want to remove. Type the number associated with the software package, then press ENTER. The VIS displays messages when the software has been removed.

Example

The following example executes the program to remove a software package from the hard disk.

```
removepkg
```

See Also

"installpkg"

restore

Name

This command restores a unit to the in-service state.

Synopsis

restore *<unit> <number> [-i] [-n]*

Description

This command is used to change the permanent state of a unit from manual-out-of-service (MANOOS) to in service (INSERV). The specified unit is placed in the INSERV state unconditionally, unless its current state is not MANOOS.

The parameters for the **restore** command are:

- *<unit>* — This parameter identifies the unit. The choices are "channel" or "card."
- *<number>* — This parameter specifies the channel or card number, a range of channel or card numbers in the form m n, or the word "all" for all the channel or card numbers. Card numbers are in the form ***card#[.port#]*** where *port#* is a port of card *card#*. If *port#* is not given, all ports of the card specified are restored. If no card number or channel number is given, a syntax message is displayed.
- *-n* — This option disables prompting from the system whether to wait until a conflict has been resolved (see the *-i* option below) or to terminate the request to restore.
- *-i* — This option is used to enable secondary command registration.If T1 diagnostics are being run, this option allows "restoring" of another card to be performed. If *-i* is used and another maintenance command is being run (**"remove", "detach", "attach", "restore", diagnose**), the request to **restore** card is blocked and a message is printed to the screen. If *-i* is not used and any maintenance command is being run, the request to **restore** card is blocked and a message is printed to the screen.

If the command is permitted to run, a check is made to see if the command is in conflict with another. A command is in conflict if the card or card associated with it:

1. Is the T1 card being diagnosed
2. Will cause a change in the existing TDM bus master assignment
3. Has an interdependency with the T1 card being diagnosed (for example, PRI)

If one of the above conflicts exist and *-n* is not used, the user is asked whether to wait until the conflict is resolved or to terminate the request. If T1 diagnostics are executing on-line tests and a conflict is detected, the **restore** command is blocked. If T1 diagnostics are executing off-line tests and a conflict is detected, the user is asked whether to wait until the conflict is resolved or to terminate the request to restore.

To delete out of the command, press DELETE. If this does not stop the command, you may need to press CTRL and backslash simultaneously. If, while running restore, you wish to abort the command, a message similar to the following may appear:

```
At the user's request, administration of the following
cmd(s) has been interrupted.

CARD NUMBERS: <card numbers>

To assure proper operation of the identified card(s),
run diagnostics at the earliest opportunity.
```

It is recommended when **restore** is aborted, diagnostics be run on all cards being administered to ensure they are returned to a fully functional state.

Example

The following example restores card 0 to service.

restore card 0

The following example restores channels 0, 1 and 5 to service.

restore channel 0-1,5

The following example restores all cards to service.

restore card all

See Also

"attach"
"detach"
"remove"

restore_appl

Name

This command restores an application.

NOTE:
This command is valid only if the Enhanced File Transfer package is installed.

Synopsis

restore_appl -n ***<application name> [-d <database file>] [-t <transaction file>] [-s <speech file>] [-p <path>]***

Description

restore_appl is used to restore an application from backed up files existing on the same machine or from backed up files sent from the host. The files are cpio files. If the file names are not specified, default file names are used and all three components (database tables, speech, transaction) are restored. The following are the default file names for each component:

database	**Dbase**
speech	**Spch**
transaction	**Trans**

The default path to restore all three components is **/tmp/sb/BkUpAppl/*<application name>***.

NOTE:
You must use the **restore_appl** command before using the **"install_appl"** command.

Return Values

If the **restore_appl** command is successful, a 0 value is returned.If any value other than 0 is returned, the **restore_appl** command failed. The following are the possible reasons for failure of the **restore_appl** command:

- The hard disk is low on space.
- You are not logged in as root or super user.
- The command syntax is incorrect.
- The command to restore the database tables failed.
- The command to restore the speech failed.
- The command to restore the transaction failed.

Example

The following example restores the application “bank_balance” from backed up files.

```
restore_appl -n bank_balance
```

See Also

"backup_appl"
"install_appl"
"remove_appl"

rmboard

Name

This command removes the hardware configuration information for a 3270 card from the system.

Synopsis

rmboard

Description

The **rmboard** command enables you remove hardware for a single 3270 host card from the system. All configuration information and the **/dev** entry is removed. No 3270 software is removed using the **rmboard** command. If you are reconfiguring a existing 3270 card, you should use **rmboard** and then **"addboard"**. This command invokes a program prompting you for all required information.

After you perform the **rmboard** command, you must execute **"stop_vs"** and **"start_vs"** from the UNIX system command line to stop and restart the voice system and deactivate the 3270 host card which was removed from the system.

You must be logged in as **root** (superuser) before using the **rmboard** command.

See Also

"addboard"
"lsboard"

save_config

Name

This command saves the **/vs/data/conf_data** to floppy disk.

Synopsis

/vs/bin/util/save_config

Description

save_config is used to save the **/vs/data/conf_data** file to floppy disk. **/vs/data/conf_data** is the file representing the configuration of a VIS machine as determined by the **/vs/bin/util/configure** program.

save_config should be used after upgrading a VIS machine in the field to store the newly determined configuration file to the CONFIGURATION DATA floppy for that particular machine.

Files

/vs/data/conf_data

See Also

"add_device"
"change_device"
"configure"
"get_config"
"remove_device"
"show_config"
"show_devices"

sb_te

Name

This command invokes the 3270 Terminal Emulator.

Synopsis

sb_te ***<session number>***

Description

The **sb_te** command is used to invoke the 3270 terminal emulator and interact as a terminal to a host. This is used to first prove that a host communications link has been established. It can also be helpful in verifying that there have not been any changes to the host application screens. Sometimes changes can occur on the host end that are not passed down to the VIS development end. The *<session number>* chosen must be released from the host interface process before invoking **sb_te**. This can be accomplished by stopping the custom data interface process (DIP) for non-Script Builder applications or by using the **"hdelete"** command for Script Builder applications.

Sessions are mapped to logical unit (LU) numbers, with sessions numbered from 0 to 31 mapped to LUs configured in card 0 and sessions numbered from 32 to 63 mapped to LUs configured in card 1. For example, session number 0 corresponds to the first LU number specified in the Configure Host Link screen for Link 0, while session number 1 corresponds to the second LU number in the Host Configure Link screen. Similarly, session number 32 corresponds to the first LU number specified in the Configure Host Link screen for Link 1, while session number 33 corresponds to the second LU number in the Configure Host Link screen, etc. A range of session numbers (for example, 5–38) can be specified to sequentially emulate each session in turn. Press CTRL Y to emulate the next session in the specified range. The CTRL Y command may only be used for multiple sessions.

If a session is not specified, the system assumes the value “all” for sessions 0–63 for both cards in a two card installation. If the first session the first card is not configured, **sb_te** automatically proceeds to the first session on the next card. For example, if session 0 on card 0 is specified and that session is not configured, a failure message is displayed and the **sb_te** command proceeds to the first session on card 1.

Example

The following example invokes the 3270 terminal emulator for card 0 and session 0.

```
sb_te 0
```

The next example invokes the 3270 terminal emulator for sessions 35–40 for card 1.

```
sb_te 35-40
```

sb_trace

Synopsis

This command displays the trace messages and the screens being sent between Script Builder applications and the 3270 host mainframe for the specified channel.

Command Format

sb_trace all

sb_trace *<voice channel number>*

Description

The **sb_trace** command displays on its stdout (usually the terminal screen) messages about what actions the Script Builder applications are executing on specified channels or sessions. It also captures screens sent between the 3270 host mainframe and Script Builder applications assigned to voice channels on the voice cards (T/R and T1) and/or sessions on the host interface card.

Once it starts running, **sb_trace** outputs the screens and trace messages that are generated from that time until the user terminates **sb_trace** by pressing DELETE. **sb_trace** outputs messages and screens when the specified voice channels or sessions are executing their Script Builder actions and/or sending or getting screens while logging-in, recovering, or in-transaction. **sb_trace** invokes the **"trace"** command when it is run.

sb_trace displays high level trace messages from the DIP and TSM on its stdout. The following is an example of the possible messages that appears:

```
Tracing started on channel 0
DIP0: CH 0 get screen form
DIP0: CH 0 save_bal =
TSM: CH 0 STEP: 0. VALUE: 10
TSM: CH 0 STEP: 1.
DB: Read Table
DB: index 0
```

The “step” refers to the corresponding action step in the transaction definition outline. “Value” refers to whatever value is given with the indicated action step.

Certain Script Builder external actions and functions may generate trace messages when they are passed invalid data or when they encounter other failures. These messages are recognizable by the fact that the “step” number is out of the range of normal action numbers that appear in the transaction definition.

If the buffer (storage area) where information is stored gets re-used before the information is completely shown on the screen, trace information may not get reported by **sb_trace**. The information you see may be incomplete. To see any missing information, place a “play message” action in the transaction to play a long silence. Insert it before the critical action whose trace you are interested in.

sb_trace accepts a voice channel as argument to output only messages and screens that relate to the specified voice channel and associated session number. For example, “sb_trace 15” limits its output to the following:

- The actions being executed from the Script Builder application on channel 15.
- If channel 15 is handling a call that interacts with the host, **sb_trace** also outputs the screen’s name, fields being sent and received, and the entire screen’s contents (24x80 bytes) for the “in-transaction” session associated with channel 15 to the file **/vs/trans/hostdata/*chan#***.
- If channel 15 is not handling a call that interacts with the host, **sb_trace** treats the channel number directly as a session number, and outputs the actions being executed and the screens sent and received by session number 15. Session 15 must be assigned to an application via **"hassign"** before **sb_trace** is invoked or else sb_trace quits without tracing. Also sb_trace only outputs the actions and screens while session 15 is “logging-in” or “recovering.”

Note that the voice channel must exist in the system or sb_trace quits without tracing.

sb_trace also accepts the keyword “all” to mean that all channels and sessions will generate output. However to trace sessions that are “logging-in” or “recovering” the corresponding session numbers must fall in the range of existent voice channels. For example, a system with 25 voice channels can only trace the first 25 sessions if they’re “logging-in” or “recovering”. The 26th, 27th, and so on sessions can only be traced when “in-transaction” and associated with a voice channel.

The screen dumps are useful for debugging Script Builder applications while they interact with the 3270 host. A file with screen dumps is created for each voice channel or session number being traced. Screens appear in chunks of 24 text lines, appended to the files in the order which they are sent or received along with their name and time of transmission or reception. They can be viewed using the standard “cat” or “pg” commands.

The files are stored in **/vs/trans/hostdata** and are named as ***chanX*** or ***chanXX***, where *X* or *XX* is a one- or two-digit channel or session number. If they exist, **sb_trace** moves these files to ***chanX*.old** or ***chanXX*.old** before starting the trace.

NOTE:
These files tend to be voluminous requiring lots of disk space. If it is necessary to remove these files, it is recommended that they be removed after stopping the voice system. Otherwise if they are removed while the voice system is running **sb_trace** stops dumping the screens until the voice system is restarted.

See Also

db_pr
db_put
"trace"

sccsDaemon

Name

This command is the daemon process which distributes messages to the SCCS or CompuLert systems and to the alarm relay unit (ARU). It also can be used to send commands from the user to the daemon process.

Synopsis

sccsDaemon *[-v NNN]*	# Run as daemon process
sccsDaemon -c {cmd}	# Send command to daemon process
sccsDaemon -h	# Print help and usage message

Description

sccsDaemon normally a continuously running daemon process which reads messages from ***${LOGROOT}*/sccsCtlPipe** where *${LOGROOT}* is normally **/usr/spool/log**. Messages are sent to this pipe by the **logdaemon** process and the message originating processes, which use the rules in ***${LOGROOT}*/ msgDst.rules** to determine where to send messages. The *-v* option sets the initial verbosity level within the daemon process. This a development debugging aid only.

As a daemon process, **sccsDaemon** has the following responsibilities:

- Logging messages received are reformatted in accordance with the MML requirements of the SCCS/CompuLert systems and transmitted to the specified device. The format is as follows:

  031 n{machine} yy-mm-dd hh:mm:ss r 033 n{pri} {process} {msd} n 031

  The control characters embedded in the message delimit the start, end, and end-of-header locations of the message.

- If output to the console is active, a copy of the message is sent to the console as well.

  NOTE:
  This form of the message is substantially different from the VIS message format and, therefore, looks different from what would appear on the console if normal message administration was used to direct

the message to the console. Use only one method to direct a message to the console.

- Every 15 minutes a “heartbeat” message is sent to the SCCS/CompuLert system, consisting only a message header. This lets the SCCS/CompuLert system know the VIS is still “alive” if there is no other message activity.
- If the alarm relay unit (ARU) is active and the priority of the message received is minor (*), major (**), or critical (*C), an alarm activation message is sent to the ARU unit to start the alarm.
- If the ARU is active and the watch dog timer is running as well, a watch dog reset command is sent to the ARU every 1 minute and 50 seconds to prevent the 2-minute watch dog timer from expiring and starting the ARU alarm.

sccsDaemon can also be used as a means to send commands to the **sccsDaemon** process. The *-c* flag indicates that it is being used to send commands to the daemon process. The commands are:

- **check_console_flag** — This command rereads the state of the console output flag. All output being sent to the SCCS/CompuLert destinations can be duplicated to **/dev/console** as well. The commands **console_off** and **console_on** use this command to inform the daemon that the state of such output has changed.
- **check_devices** — This command rereads the destinations for output to the SCCS/CompuLert systems and for the ARU. The command **assign_tty** uses this command to alert the daemon process that the destinations have been altered.
- **quit** — This command causes the daemon process to exit gracefully.
- **verbosity {value}** — This command changes the verbosity level of the daemon process. There are some messages that are only logged when the verbosity level is increased. This is a development debugging aid only.

The **sccsDaemon** process takes option settings from **/vs/etc/default/sccsDaemon**. The following parameters are understood:

CONSOLE_OUTPUT_DEFINED

If this parameter is set, messages are sent to the **sccsDaemon** can be sent to **/dev/console** in addition to the SCCS device. The default for this parameter is FALSE.

CONSOLE_FLAG

This parameter enables output to console if CONSOLE_OUTPUT_DEFINED. Its value can be changed during operation via the **console_on** and **console_off** command. The default for this parameter is FALSE.

CONSOLE_DEVICE
: This parameter specifies the name of the console device. The default for this parameter is **/dev/console**.

ARU_ENABLED
: This parameter enables the sending of alarm messages to the ARU. The default for this parameter is TRUE.

SCCS_DEVICE
: This parameter specifies the alternate device name for the SCCS. This parameter is only used if the file **/vs/data/Sccs_tty** is not set. The default for this parameter is <NONE>.

ARU_DEVICE
: This parameter specifies the alternate device name for the ARU. This parameter is only used if the file **/vs/data/Aru_tty** is not set. The default for this parameter is <NONE>.

SCCS_PIPE
: This parameter specifies the name of the **sccsDaemon** pipe. The default for this parameter is **/usr/spool/log/sccsCtlPipe**.

MACHINE_NAME
: This parameter specifies the alternate machine name. This parameter is only used if the file **/vs/data/Machname** is not set. The default for this parameter is <NONE>.

VERBOSITY
: This parameter sets the initial value of the internal verbosity flag for the **sccsDaemon**. The default for this parameter is 0.

SCCS_MODES
: This parameter uses the stty command to condition the SCCS device. The default for this parameter is **stty sane 9600 erase b echoe echok**.

ARU_MODES
: This parameter uses the stty command to condition the ARU device. The default for this parameter is **stty sane 9600 erase b echoe echok**.

Files

/vs/bin/vrs/sccsDaemon	# Executable program
/vs/data/Sccs_tty	# Contains SCCS/CompuLert device
/vs/data/Aru_tty	# Contains ARU device
/vs/data/Machname	# SCCS/CompuLert name of system
/vs/data/console_stat	# Current status of output to console
/vs/data/wdog_stat	# Current status of ARU watch dog timer
/usr/spool/log/sccsCtlPipe	# Name of daemon pipe

See Also

See the *CONVERSANT VIS CompuLert/SCCS*, 585-350-808, for additional information about the following commands.

assign_tty
chg_machname
console_off
console_on
wdog_off
wdog_on

show_config

Name

This command displays and prints to file the valid CONVERSANT VIS system configuration represented by the **/vs/data/conf_data** file or the incomplete configuration represented by the **/vs/data/fail_data** file.

Synopsis

***/vs/bin/util/show_config** [fail | filename]*

Description

The output of the **/vs/bin/util/configure** program is either the **/vs/data/conf_data** file or the **/vs/data/fail_data** file depending respectively on whether the program arrives at a complete and valid configuration, or an incomplete, conflicting configuration based on the user's input. In either case, the data in these files is compressed and cannot be easily understood. The **/vs/bin/util/show_config** command formats the data in these files and displays it in tabular form to the screen and also writes it to a file in the current directory.

A message from the **"configure"** program instructs the user whether to use the *fail* option with **show_config** or not.

If the **"configure"** program was successful, executing **show_config** with no argument creates a **./configuration** file, by expanding the contents of **/vs/data/conf_data**. This file can then be printed for hard copy of the successful configuration.

If the **"configure"** program was unsuccessful at determining a configuration, executing **show_config** fail creates a **./failed_config** file by expanding the contents of **/vs/data/fail_conf**. The **./failed_config** file may be examined to determine what conflicting resource caused the configuration to fail.

show_config always checks for the presence of **./configuration** or **./failed_config** and asks the user whether it is acceptable to overwrite the current file by that name if it exists.

When the **"configure"** program is used to upgrade an existing machine, the current **/vs/data/conf_data** file is saved in **/vs/data/conf_*MMDDYY***, where *MM* = month, *DD* = day and *YY* = year. It may be desirable at times to see what configuration is represented by these saved configuration files. **show_config** may be used to expand the contents of a saved configuration file by specifying the filename as the first argument. The user is prompted for an output file name whenever the first argument is an input configuration filename.

Files

/vs/data/conf_data
/configuration (show sample output, describe each HEADING)
/vs/data/fail_data
/failed_config

See Also

"add_device"
"change_device"
"configure"
"get_config"
"remove_device"
"save_config"
"show_devices"

Notes

The **"show_config"** command takes only one argument. The *fail* and *filename* arguments are mutually exclusive.

show_devices

Name

This command displays and prints to file all devices and their attributes as represented in the **/vs/data/device_data** file.

Synopsis

/vs/bin/util/show_devices

Description

The **/vs/bin/util/show_devices** command uncompresses the database of devices and their attributes contained in the **/vs/data/device_data** file and displays the information to the screen. At the same time, a **./devices** file is created so that hard copy of this information may be generated. If a **./devices** file already exists, the user is prompted as to whether it is acceptable to overwrite the file.

Files

/vs/data/device_data
./devices (show sample output)

See Also

"add_device"
"change_device"
"configure"
"get_config"
"remove_device"
"save_config"
"show_config"

soft_disc

Name

This command sends a disconnect to a script on a channel or channels.

Synopsis

soft_disc *<channel>*

soft_disc *<channelStart-channelEnd>*

Description

soft_disc sends a message or messages to TSM requesting that the script running on *<channel>* or the range of channels *<channelStart-channelEnd>* be sent interrupt messages. If no script is running on the channel or if TSM does not own the channel, no action is taken for the channel.

soft_disc waits for a response from TSM. When it exits, TSM has acted on all the requests for all the channels by sending disconnects to the scripts or rejecting the requests. Scripts running on the channel receive the ESOFTDISC event.

Return Values

If the **soft_disc** is successful, a 0 value is returned. If any other value than 0 is returned, the **soft_disc** command completely or partially failed. If **soft_disc** returns a value of 2, then **"dip_int"** command failed due to temporary condition. In this case, the user should attempt the **"dip_int"** command again.

Example

The following example requests that TSM send interrupt messages to channel 2.

```
soft_disc 2
```

The following example requests that TSM send interrupt messages to channels 1 through 32.

```
soft_disc 1-32
```

See Also

"dip_int"

soft_szr

Name

This command starts a script on a channel.

Synopsis

soft_szr ***<channelStart-channelEnd> <script>***

Description

The **soft_szr** command can be used to start a script on a channel. The **soft_szr** command sends a message to TSM requesting that a script be started on a channel. If the channel is in use, the script is not started. **Soft_szr** waits for a response from TSM. When **soft_szr** exits, TSM has either accepted the request and started the script or rejected the request.

There are two arguments to the **soft_szr** command, *<channel>* and *<script>*. The *<channel>* argument specifies the channel or range of channels on which you want to start the script. The *<script>* argument specifies the script to be started. The script does not have to be in the table of assigned scripts.

The channel number(s) must be valid and the channel(s) must not be busy, and the channel(s) must be in the inserv state. If you specify a channel that is busy, the command fails. If you specify a range of channels and one or more of the channels is busy, the command seizes the idle channels but fails for the busy channels.

Example

The following example starts the script “sodapop” on channels 0 through 4.

```
soft_szr 0-4 sodapop
```

The following example starts the script “test1”’ on channel 10.

```
soft_szr 10 test1
```

Return Values

If the **soft_szr** is successful, a 0 value is returned. If any value other than 0 is returned, the **soft_szr** command completely or partially failed. If **soft_szr** returns a value of 2, then **soft_szr** command failed due to temporary condition. In this case, the user should attempt the **"dip_int"** command again.

spch2unix

Name

This command converts speech slices into UNIX file systems.

Synopsis

spch2unix *-n <name> -d <raw_disk_slice>*

Description

The **spch2unix** command converts speech slices into UNIX file systems. **spch2unix** converts the speech slice or partition, specified with the *-d* option as *<raw_disk_slice>*, into a UNIX file system mounted as ***/<name>*** where *<name>* is the argument to the *-n* option. The *-n* and *-d* options are required.

Example

The following example converts the speech slice found on **/dev/rdsk/1s3** into a UNIX file system mounted as **/usr3**.

```
spch2unix -n usr3 -d /dev/rdsk/1s3
```

Warnings

All speech data on the converted speech slice is lost when this command is used. **spch2unix** should never be run on non-speech slices.

spCtlFlags

Name

This command sets and clears flags used to control the behavior on SP Executive pack files as they run on an SP card.

Synopsis

spCtlFlags ***[-b SP-index] [-t] [[+|-]flag]***

Description

The CTL flags provide a means to alter the behavior of code running on the SP from the PC without distracting it from the job at hand. At the current time the CTL flags integer is divided into three parts, the upper 16 bits, which are general purpose flags to be used to turn on and off code and **printfs**, the bottom 8 bits, which are reserved for the SP Executive functions, and bits 8 15, which are currently not used by anyone officially. There is an unofficial use of these bits to prime the verbosity level for layer 3 of PRI.

The following are the options that can be used with the **spCtlFlags** command:

-b SP-index	Index of the SP card to be examined
-t	Terse - only output hex value of flag

NOTE:
spCtlFlags only works with SP executive applications (currently, the PRI and CCA pack files).

With no flag argument, **spCtlFlags** just prints the current value. With a flag argument, it either resets the value (no ‘+’ or ‘-’), logically ORs in the flag (‘+’), or logically and compliments out the flag (‘-’). A flag can either be a number or use one of the following symbolic names:

printf	This flag controls whether printfs from within an SP card actually generate output or not.
letters	This flag contains executive trace flag of letters arriving from the PC.
terminations	This flag generates reports on all process and action terminations.
dbgpanics	If this flag is set, panics by SP executive go to debugging monitor. If not set, panics go immediately to ROM for reloading.
timefcns	This flag enables timing of TDM and DSP functions.
checkmem	This flag enables checking of the “malloc” arenas to insure that they have not been corrupted. (This is fairly expensive in terms of CPU cycles expended per allocation reference.)
enabledbg	This flag enables various general purpose debugging code if it is compiled into the executive.
dbg{1-16}	This is a general purpose flag that can be used for debugging.

Symbolic and numerical flags can be combined with the “+” sign between them, that is, “+dbg1+printf” or “-0x20+printf.”

Notice that the current value of the flags is printed if no other arguments are specified and that by starting the flags with a ‘+’ causes them to be added to those already in place rather than just replacing the current flags with the new ones. The following is additional information for each of the symbolic names:

printf	If this flag is not set, all **printf()** operations from within the SP Executive are essentially NOPs. This flag must be set for any print information to be sent to the PC and logged.
letters	If this flag is on, the SP Executive attempts to report, via **printf()** the arrival of each letter that it is processing from the PC.
terminations	If this flag is on, the SP Executive sends a termination letter to the SP whenever a process or an action completes. This, in turn, is logged.
dbgpanics	If this flag is set and the SP Executive calls the **panic()** routine, it stops and waits for a debugger to examine what has happened. If this flag is not set and **panic()** is called, the SP Executive returns immediately to the ROM for reloading.

timefcns	If this flag is set, the SP Executive starts timing operations on each of the following four things, TDM interrupt servicing, the length of time between TDM interrupts, the length of time DSP loading is taking, and the length of time DSP servicing is taking. This information is requestable in the future via a letter from the PC. Currently it must be examined via a debugger.
checkmem	If this flag is set, each attempt to **malloc()**, **realloc()**, or **free()** memory causes the malloc arena to be checked for consistency. If the define symbol SM_FULLCHECK is set when the **spaceMngr.c** file is compiled, this check is very complete (though more time consuming) and detects problems sooner. If it is compiled without SM_FULLCHECK, the check is more cursory in nature.
enabledbg	Much of the special history keeping code is conditional upon this flag being set. If it is not set, the overhead of saving and timing is avoided. If it is set, then whatever history mechanism has been compiled in, saves its form of history information for debugging purposes.
dbg[1-16]	The use of these flags is up to each task. It is assumed that they will be used during debugging phases, but not be in use for final distribution. Code using them does tests of the following form:

```
if (spcon->status[SPS_CTL_FLAGS] & SPCF_DBGnn)
```

to determine whether a certain section of code or not should be executed.

spres

Name

This command restores speech from a backup.

Synopsis

spres *-I <file> [-v] -t [talkfile <list>] [phrase <list>] [listfile <list>]*

Description

The **spres** command restores the specified talkfile number, phrase number, listfile, or phrase and talkfile of the speech. Only speech that is backed up using the **"spsav"** command can be restored with the **spres** command.

The arguments for the **spres** command are:

-I file	This argument specifies the input device. Typically, this is cartridge tape (**/dev/rmt/c0s0**).
-v	This argument is the verbose flag that gives running commentary of restore procedure.
-t	This argument is the tape flag. This is required for restore from cartridge tape.
[talkfile<list>]	This argument specifies the list of talkfiles to be restored, specified as a single digit, a range m–n, or the word “all.” If no value is given, the default is “all.”
[phrase<list>]	This argument specifies the list of phrases to be restored, specified as a single digit, a range m–n, or the word “all.” If no value is given, the default is “all.”
[listfile<list>]	This argument specifies the list of listfiles and associated speech to be restored (for example, listfile list.cabnt).

The **spres** command invokes an interactive program asking you to you insert and remove cartridge tapes periodically. If the *-v* option is used, the system displays information about each step of the recovery.

Example

The following example restores listfile “list.cabnt” verbosely from cartridge tape.

```
spres -l /dev/rmt/c0s0 -v -t listfile list.cabnt
```

spsav

Name

This command backs up speech.

Synopsis

spsav -O ***<file> [-v] -t [talkfile <list>] [phrase <list>] [listfile <list>]***

Description

The **spsav** command backs up the specified talkfile number, phrase number, listfile, or phrase and talkfile of the speech. Only speech in the speech file system can be backed up using the **spsav** command. The arguments for the **spsav** command are:

-O file	This argument specifies the output device. Typically, this is cartridge tape (**/dev/rmt/c0s0**).
-v	This argument is the verbose flag that gives running commentary of speech being saved.
-t	This argument is the tape flag. This is required for back up to cartridge tape.
[talkfile<list>]	This argument specifies the list of talkfiles to be backed up, specified as a single digit, a range m–n, or the word “all.” If no value is given, the default is “all.”
[phrase<list>]	This argument specifies the list of phrases to be backed up, specified as a single digit, a range m–n, or the word “all.” If no value is given, the default is “all.”
[listfile<list>]	This argument specifies the list of listfiles and associated speech to be backed up (for example, listfile list.cabnt).

The **spsav** command invokes an interactive program asking you to you insert and remove cartridge tapes periodically. If the *-v* option is used, the system displays information about each step of the back up.

Example

The following example saves listfile “list.cabnt” from cartridge tape.

```
spsav -O /dev/rmt/c0s0 -t listfile list.cabnt
```

spStatus

Name

This command displays information about the pack file running on an SP card.

Synopsis

spStatus *[-b SP-index] [-i interval] [-c count] [-r] [-B]*

Description

A substantial amount of information about the state of an SP Executive pack (PRI and CCA pack files) is available via shared memory and the program **spStatus**, which displays the information. The information is defined in **include/spStatus.h.**

The following are the options that can be used with the **spStatus** command:

-b SP-index	Index of the SP card to be examined
-i interval	Interval between examinations of SP status Minimum: 2 seconds. Default: 60 seconds.
-c count	Number of times SP status is to be examined. Default: 1
-*-r*	Reset the executive and task counts before starting.
-B	No bell when running in iterative mode.

spStatus can be run in a one-shot mode, which is the default, or an iterative mode. In the iterative mode, it prints the changes between each successive examination of the values stored in the **spcon** structure in shared memory.

Sample Format

The following is an example of the sample output if **spStatus** is against the CCA pack.

```
           Fri Dec 7 13:06:03 1990
Romstate: 0x0 Romcmd: 0x0 Romargs: 0x0 0x0
Ramstate: 0x245 Pack Features: C      Pack Type: SP executive
 Bootcnt: 0x0 SPtime: 0x6a5 SPusage: 0x0
Debug ID: 0 spFreeMemory: 1,164,152
<< Status Information >>
            Free Actions: 46
            Busy Actions: 4
          Active Letters: 4
               Free DSPs: 2
             Broken DSPs: 0
               Busy DSPs: 0
        Run Queue Length: 0
      Sleep Queue Length: 4
      Running Process ID: 5
    Running Action Index: 3
            DSP Requests: 4
       RPC Requests Done: 0
     RPC Requests Queued: 0
    RPC Requests Discard: 0
             Exception #: 0x0 0-Reset
           Exception Adr: 0x0
                 Routine: 0x0
     PC at last TDM Intr: 0x99f05b62
     PC at last DSP Intr: 0x99f05b46
               DSP Count: 2247
               CTL Flags: 0x0
          Timer Requests: 0
           Active Timers: 0
        Completed Timers: 0
           Killed Timers: 0
Work Search Loops * 1000: 53
            TDM Overruns: 0
 TDM Servicings Deferred: 0
        Letters Received: 8
            Letters Sent: 0
        Letters Deferred: 0
       Letters Discarded: 0
          Executive Time: 57
               Idle Time: 489
Task[6]: 1156
<< Mailbox Information >>
```

```
Index         1st Empty   1st Full
0 -> PC        00
1 <- PC        00
2 <- PC        4040
3 <- PC        00
4 <- PC        00
5 <- PC        00
6 <- PC        00
7 <- PC        00
8 -> PC        00
====================================================
1          Fri Dec 7 13:06:09 1990
 Bootcnt: 0x0 SPtime: 0x7df SPusage: 0x0
<< Status Information >>
        Run Queue Length: 1(+1)
      Sleep Queue Length: 3(-1)
      Running Process ID: 4(-1)
    Running Action Index: 2(-1)
     PC at last TDM Intr: 0x99f14024
     PC at last DSP Intr: 0x99f08154
               DSP Count: 2710(+463)
Work Search Loops * 1000: 57(+4)
          Executive Time: 58(+1)
               Idle Time: 537(+48)
Task[6]: 1421
```

Following is a brief description of each element of the display:

Romstate, Romcmd, Romargs
These three values are active if either the ROM is in control of the SP card or a debugger is in charge.

Ramstate, Pack Features, Pack Type
If a packfile is running or being debugged Ramstate contains the ID of the pack. If the pack is an SP Executive type pack, the Pack Features indicate which tasks are available in this pack. The Pack Type is either “SP executive” or “Original.”

Bootcnt, SPtime, SPusage
Bootcnt is incremented each time the ROM restarts. Only diagnostics currently alter it in any other way.   SPtime is the time in 16 msec increments since the pack started. If **spStatus** is running in recursive mode, this value is not changing, and the debugger is not active, a warning, `“NO Clock! Check TDM master,”` is generated. One of two things is happening, either there is no TDM master and hence no TDM interrupts, or the pack file is stuck at priority level 6 or 7 and so all interrupts are blocked. In the former situation, check your T1 or Tip/Ring (T/R) cards and make sure that one of them is the TDM master. In the latter case, you have a bug. Use msdb and examine the

pack file. **SPusage** is the current load factor on the SP card. This is the last value of meaning if the pack is an original-style pack. The remaining information applies only to SP Executive packs.

Debug ID, spFreeMemory
Debug ID is set to the pid of the UNIX process currently debugging this SP card. It is used to avoid collisions between people attempting to debug code running on a card. **spFreeMemory** is the amount of memory free in the memory allocation arenas, which are managed by **malloc()**, **realloc()**, and **free()**.

Free Actions
The number of **Action** structures not currently assigned to a time slot. This value is initially 50.

Busy Actions
The number of **Action** structures currently assigned to time slots.

Active Letters
The number of letters being carried in **Chainmail** structures for long time processing via **Action** structures.

Free DSPs, Broken DSPs, Busy DSPs
The number of DSP processors available to do work, broken, and assigned to work.

Run Queue Length, Sleep Queue Length
The number of processes waiting to run and the number waiting for some event to wake them up.

Running Process ID, Running Action Index
The process ID of the SP Executive process currently running and the index of the **Action** structure currently active.

DSP Requests
The number of **DspRequest** structures active.

RPC Requests Done, RPC Requests Queued, RPC Requests Discard
The number of remote procedure call requests that have been performed, the number that are waiting to be done, and the number of requests that had to be discarded before the backlog was too large.

Exception #, Exception Adr
The 680X0 hardware exception number and the name of the exception that has stopped the 680X0 processor and either sent it to the ROM or to the debugger and the address where the exception took place. These can be very valuable in case of a fatal error.

Routine
Currently not used.

PC at last TDM Intr, PC at last DSP Intr
Addresses at which the TDM and DSP last interrupted.

Info Flags
Currently there are two pieces of information conveyed by these flags, whether the processor is currently within a DSP interrupt and whether it is within a TDM interrupt. Both, neither, or any combination could be true.

DSP Count
The number of DSP interrupts processed.

CTL Flags
The current value of the CTL flags. These are used to control optional code within a pack. See **spCtlFlags** for further information.

Timer Requests
The number of timer requests that have been made.

Active Timers, Completed Timers, Killed Timers
The number of timer requests currently outstanding, the total number of timer requests that have run to completion, the number of timer requests that were removed prior to execution. If these values do not total up properly, there is also a warning indicating that there is trouble.

Work Search Loops
This is the number of times divided by 1000 that the SP Executive has gone through its base level work search loop, trying to find something productive to do. The change in the number goes down as the SP Executive becomes busier and busier doing productive work.

TDM Overruns
This number should always be zero. If it is not, it indicates that some activity is taking too long and blocking the processing of a TDM interrupt before it rolls over and starts overwriting data. This is serious.

TDM Servicings Deferred
This is the number of times that a TDM servicing was deferred because the TDM interrupt came in on top of a DSP interrupt for a time slot. It is not serious. It just indicates that the hardware is busy and conflicts are being resolved. It can be a potential area of difficulty if the DSP routine is too slow and the TDM overruns while it is waiting to be serviced.

Letters Received, Letters Sent, Letters Deferred, Letters Discarded

This is the number of letters received from the PC, the number of letters sent to the PC, the number of letters going to the PC that had to be temporarily stored in the overflow area because the PC was not keeping up, and the number of letters that even the overflow area could not handle and had to be discarded. Going into the overflow area is an indication of potential trouble, but is not bad if the duration is short. If the SP code continues to generate too many letters in too short of a period of time, then it is real trouble. The same thing can happen if the PC gets bogged down and cannot keep up.

Executive Time, Idle Time, Task[]

These counts indicate the load being placed on each portion of the system. The executive time is the number of times the TDM interrupted some activity of the SP Executive that was what is considered to be the idle look-for-work activity. The idle time is the number of times the TDM interrupted the look-for-work activity. When tasks are active, a line appears for each task. The index of the task is its position in the **tasks[]** array found in the associated **sp/config/taskTbl*.c** file.

Mailbox Information — Index, Empty, Full

The mailbox information is rudimentary information about activity within each mailbox. It does not tell you how many letters have been sent via each mailbox, though that may come eventually, but it does tell you whether the mailbox is empty (1st Empty == 1st Full) and if the values are changing from one display to the next, you know mail is passing through that mailbox. Keep in mind that mailbox 1, from the PC, is now reserved by the kernel and is used by the ioctl() form of mail sending for all processes other than the limited number of processes that directly own mailboxes.

In iterative mode, only those lines whose values have changed since the list display are listed. On decimal entries, the delta value since the last time is also printed.

spVrsion

Name

This command prints the version of the SP driver currently installed on a machine.

Synopsis

spVrsion

Description

The **spVrsion** command prints which version of the SP driver has been installed. The two versions that can be installed are the 12-Mbyte version and the 44-Mbyte version.

start_vs

Name

This command brings the system up to a fully operational state.

Synopsis

start_vs

Description

This **start_vs** command returns the voice system software to fully operational state. If you use the **"stop_vs"** command to stop the system, you should use the **start_vs** command to start it again. **start_vs** also should be used if the system was rebooted or powered down after **"stop_vs"** was used.

The **start_vs** command checks to see if the user stopped the system with the **"stop_vs"** command. **start_vs** places all cards placed in the manual-out-of-service (MANOOS) state with the **"stop_vs"** command in the in-service (INSERV) state.

You must be logged on to the system console as **root** to use the **start_vs** command.

Since the **/vs/data/spchconfig** file cannot be edited while the voice system processes are running, it is a good idea to check the value of nbufs in the **/vs/data/spchconfig** file before executing the **start_vs** command. The value of nbufs defines the number of speech buffers. In order for the voice system to operate properly, nbufs must be set to 2.5 times the number of active channels.

Example

The following example starts the voice system software.

```
start_vs
```

See Also

"stop_vs"

stop_vs

Name

This command gracefully stops the voice system software.

Synopsis

stop_vs ***[time_out] [-n]***

Description

The **stop_vs** command gracefully stops the voice system software. If the system is receiving calls, **stop_vs** waits for approximately three minutes before it unconditionally stops the software. By waiting, the system allows callers to finish their transactions. **stop_vs** disables incoming call recognition on all cards to prevent them from being reactivated by an incoming call.

The *time_out* option is the time to wait before the voice system is stopped. The default value for this option is 180 seconds. The *-n* option prompts you with a message that another maintenance command (**"restore"**, **"remove"**, **"attach"**, **"detach"**, **diagnose**) is being performed. It asks you if you wish to continue or to terminate the **stop_vs** command. The **stop_vs** command terminates another maintenance command in progress when initiated. The default value for this option is Yes.

If you use **stop_vs** to stop the system, you should use **"start_vs"** to reactivate it. If you use **stop_vs** to stop the software and then reboot the machine, be sure to execute **"start_vs"** after logging in as **root**. This ensures that the system is returned to the state it was in before it was rebooted.

If an active host link is established, the **stop_vs** command checks the LUs and logs out the application(s). The command waits up to 60 seconds (6 series of the 10 seconds each), then continues stopping the voice system.

Example

The following example stops the voice system software.

```
stop_vs
```

See Also

"start_vs"

sysmon

Name

This command executes a program that monitors incoming telephone lines and the associated cards to see that they are functional.

Synopsis

sysmon *<page number>*

Description

The **sysmon** command verifies that each incoming telephone line and its associated card are functional. Before initializing the test, locate a touch-tone telephone close to the system controller and get a telephone number to be used for dialing into the system. Use the **"assign card/channel"** command to assign to a group any channels you want to test. Then, use the **"assign service"** command to assign a script to the same group.

Once the channels and service are assigned, enter the **sysmon** command followed by the number of pages, or screens, you want to see. Each page displays 120–140 channels.

The resulting display shows all channels and their current states. Note that only equipped channels can be in the IDLE or MOOS state, while unequipped channels are followed by dashes (--).

Enter the telephone number for the touch-tone phone. Watch the display on the monitor and note the channels that receives the call. Follow the instructions provided by the voice system. Enter 0000 to end the test.

Example

The following example shows page four of the system monitor display.

sysmon 4

tas

Name

This command executes the transaction assembler (tas) program to assemble script instructions.

Synopsis

tas *[-e] [-I<include_directory> -T<talk_directory> -U<name> -D<name> -D<name_def> -Y<dir> -H] -o<output_file> <application_name>*.t

Description

Tas is used to assemble script instructions recorded in an ***application-name*.t** file. It produces an executable file designated ***application-name*.T**, which is stored in a table as a list of executable script instructions.

The *-e* option requires exact string matches for speech phrases.

The arguments must be in the order given above for the command to work properly. The directory search specified by the arguments are: *I* (include file) and *T* (listfile). No space is allowed between the *-I* and *-T* flags and their pathnames, but space is allowed after the *-e* flag. Note that the *-I* option to **tas** is interpreted by cpp(1).

The remaining arguments are:

- *-U <name>* — Remove any initial definition of name, where name is a reserved symbol that is predefined by the particular preprocessor (this option is interpreted by cpp(1)).
- *-D <name>* and *-D <name-def>* — Define *name* with value *def* as if by a #define. If no *-def* is given, *name* is defined with value 1. The *-D* option has lower precedence than the *-U* option. That is, if the same name is used in both a *-U* option and a *-D* option, the name is undefined regardless of the order of the options (this option is interpreted by cpp(1)).
- *-Y <dir>* — Use directory *dir* in place of the standard list of directories when searching for #include files (this option is interpreted by cpp(1)).
- *-H* — Print, one per line on standard error, the path names of included files (this option is interpreted by cpp(1)).
- *-o <output_file>* — The name of the output file.

The default output for this instruction is **out.T**.

Note that the maximum number of literals per script allowed by the **tas** command is 450. If there are more than 450 literals in a script, the error message “literal table overflow” is displayed. Additional limitations enforced by the **tas** command are (whichever occurs first in a list file):

— 1,000 phrases

— 4,000 words

— 40,000 characters

If more phrases are needed by an application, use multiple list files and tfile instructions within the script.

NOTE:

If your script contains a large number of define statements, **tas** may report messages such as the following during compilation:

```
script.t: 1068: too much defining
```

where *script.t* is the script source file and *1068* is the line in which the define appears. The limit to the number of define statements that a script may have depends on the number of defined macros and their size. If this type of message appears, reduce the number of define statements in your script.

Files

/vs/bin/tas

Example

tas example.t

The program includes applicable header files and replaces literal definitions with corresponding numbers to produce an assembled version of the script. The assembled code is stored on disk under the label example.T. The unassembled instructions are found in the file **/var/applN/trans/example.t**.

tas example.t -I/var/include -T/var/speech

In addition to performing the same functions described for the previous example, tas checks the files in **/var/include** when processing include statements and the file in **/var/speech** when processing T-file statements.

trace

Name

This command outputs trace messages for the specified processes and channels.

Synopsis

trace ***[name]...[ch <range>]...[card <card #[port#>]]...[sleep <interval>]***

Description

Trace displays on standard output (stdout) the lines (trace messages) written into the VIS trace buffer after the trace command begins to run. Note that processes typically write to the trace buffer only after trace has started. Once trace is started, it continues indefinitely to display any new trace message that is written into the buffer. Press DELETE to stop the tracing.

The *name* arguments specify the processes that have their trace messages written to the trace buffer. This leaves the limited internal trace buffer space available for tracing the selected processes. However, this control only applies to trace messages that use the **db_put** library function. Using **db_put** instead unconditionally writes out the trace message regardless of what processes are being traced. Typing trace without any arguments lists the current processes that are posted in the bulletin board and can be traced.

Likewise, tracing can be done on a specified number of channels. Trace messages specified using dp_ch are written to the trace buffer if they correspond to the channels being traced. The channels to trace is specified in one of two ways:

1. Explicitly by using the chan argument
2. Indirectly by specifying the card and optional port numbers

When there are no longer new trace messages to be printed, trace falls asleep for the specified interval (in milliseconds). The default for sleep is 200 milliseconds.

Upon successful completion, **trace** returns an exit status of zero. Otherwise, a diagnostic message is displayed on standard error (stderr) and one of the following exit codes is returned:

1	The voice system is not running or the specified sleep interval is negative or an invalid argument is specified.
2	Can not attach the bulletin board (BB)

NOTE:
At times, trace loses messages when the trace buffer fills up and overflows, meaning trace could not display the messages fast enough. As a result, trace displays the number of messages lost. This usually happens when too many processes and/or channels are being traced or when the sleep interval is too long.

Setting the sleep interval to zero or a small number might affect performance adversely as trace is continually using the central processing unit (CPU). Only one trace should be running at any one time in the system. Otherwise, the multiple traces contend and interfere with each other as they try to read the trace buffer.

Examples

The following example traces TSM, ET, and channels 0-5:

```
trace tsm et ch 0-5
```

The following example traces channels 0-5 plus 15, 17, and 19 with a sleep interval of 30 seconds:

```
trace ch 0-5,15,17,19 sleep 30000
```

or

```
trace ch 0-5 ch 15,17,19
```

The following example traces the channels on card 2 and on card 2 port3 and channel 35:

```
trace card 2 card 2.3 ch 35
```

See Also

db_pr
db_put

trarpt

Name

This command generates a call traffic report.

Synopsis

trarpt *<hours> <summarize> <date>*

Description

The **trarpt** command generates a call traffic report. Information in this traffic report includes the number of calls coming in to the system during a specified time period, average holding time, and the percentage of time the channel was occupied for a certain hour. This report is stored in standard out (stdout). Before this can be done, the database system must be up and running, but the voice system does not need to be up.

The parameters for the **trarpt** command are:

- *<hours>* — This parameter specifies the hours in which the traffic data was collected. The valid options can be a range between 0 to 23 (with 0 representing midnight and 23 representing 11 p.m.), or "all."
- *<summarize>* — This parameter indicates a traffic report or a traffic summary report to be generated. If the option is "n", the report provides information on the total traffic volume for each channel in one-hour increments. If the option is "y," the report is a summary report that provides information on the total traffic volume for each channel for the whole period specified in the *<hours>* parameter.
- *<date>* — This parameter specifies the date the data was collected in the system. This parameter must be in the mm/dd/yy format.

Example

The following example generates a traffic summary report for data collected on date August 24, 1993 between 8 a.m. and 5 p.m. on multiple entries per channel.

```
trarpt 8-17 y 08/24/93
```

The following example generates a traffic report for data collected on date August 24, 1993, one entry per channel)

```
trarpt all n 08/24/93
```

upg

Synopsis

This command provides automated assistance in upgrading a CONVERSANT VIS machine to Version 4.0.

Command Format

***/usr/lib/upgrade/upg** [-v] [-f] [-t] [-d] [-c] [-F] [-h]*

Description

The **upg** command and all of its associated files and functions are installed as part of the Software Upgrade Assistance Package delivered as part of an upgrade to CONVERSANT VIS Version 4.0. The command assists with the procedure of upgrading a CONVERSANT VIS machine with Version 3.0 or later software on a MAP platform to a Version 4.0. It provides online guidance in implementing the upgrade procedures. In combination with the *CONVERSANT VIS Version 4.0 Software Upgrade*, 585-350-111, the tool simplifies the upgrade procedure and eliminates many sources of potential errors in upgrades.

The procedures implemented by this command are intended to be self-documenting. Read and carefully follow all instructions which appear on the screen during the upgrade.

Before executing the command, perform all the actions defined in “Getting Started” in Chapter 1, “Introduction to Software Upgrades,” of *CONVERSANT VIS Version 4.0 Upgrade*s, 585-350-110, which applies to the particular customer scenario being upgraded. Help is available for most requests made by the upgrade package by typing “?.” Type “q” to exit the upgrade procedure at any of these prompts. Re-entering the command resumes the upgrade procedure from where it was interrrupted.

The following options are supported with the **upg** command:

- *-v* — This option runs the command in verbose mode, printing a considerable amount of output both to the screen, and at the user’s option, to a file.
- *-f* — This option specifies the CONVERSANT VIS software release running on the machine being upgraded *before* the upgrade procedure has taken place. Normally, this option need not be specified. The upgrade tool determines this information by determining the version of the CONVERSANT VIS Application Software package running on the source machine.

- *-t* — This option specifies the CONVESANT VIS software release to which the machine is being upgraded.
- *-d* — This option forces the ORACLE database and associated SQL packages to be removed and reinstalled even if they are compatible to the software release to the machine that is being upgraded.
- *-c* — This option introduces an additional confirmation step between each of the phases of the upgrade procedure, allowing you to abort the upgrade procedure (by entering '**q**') between each phase.
- *-F* — This option forces all packages to be loaded from floppy, even if the packages are available on tape.
- *-h* — This option prints a help message defining the **upg** command syntax.

Notes

Help messages and graceful user exit from the questions which are part of each software package's individual removal or [re]installation are not available at this time.

Files

All files associated with the Software Upgrade Assistance tool are stored/written in the **/usr/lib/upgrade** directory during the upgrade procedure. Save these files if you choose to remove the Software Upgrade Assistance package from the system before verifying *all* functionality of the upgraded system. If an upgrade includes a disk change of any kind (disk upgrade, repartition, etc.), then the first three files should be saved to floppy disk before any modifications are made to the disk.

- **output.list** — This file is a record of most screen interactions during the entire upgrade procedure.
- **CVIS.info** — This file is a survey of some hardware and software parameters and configuration settings on the system *before* it is upgraded. This file contains most of the information needed to install or reinstall newer versions of the software packages on the system.
- **rmvPkgsList** — This file contains an ordered list of the "older" software packages on the system. These are the packages that need to be removed and/or installed or reinstalled on the system to accomplish the upgrade.
- **insPkgsList** — This file contains an ordered list of the "new" software packages to be installed to accomplish the upgrade. These are the packages that need to be installed or reinstalled on the system to accomplish the upgrade.

- **fileList** — This file contains a list of the files encounted on the system which will be preserved in some form during the upgrade procedure. Many, but not all, will be restored on or merged into the target system during the upgrade.
- **ignoreList** — This file contains a list of files which are to be ignored, even if a later operation indicates they should be preserved.

vdf

Name

This command lists the number of free 16-kbyte blocks on each speech disk slice.

Synopsis

vdf

Description

NOTE:
The VIS must be running to execute this command.

The **vdf** command displays the number of free 16-kbyte blocks on each disk slice listed in the **/vs/data/fslist** file. The mode (readwrite/readonly) also is displayed.

Example

vdf

vsdisable

Name

This command disables the automatic restarting of the voice system.

Synopsis

vsdisable

Description

The **vsdisable** command is used to prevent the voice system from being started when the CONVERSANT VIS is rebooted. Running **vsdisable** allows you to log into the system before the voice system is started. The voice system may be started manually at any time with the **"start_vs"** command.

Example

vsdisable

See Also

"vsenable"

vsenable

Name

This command enables the automatic starting of the voice system at system reboot.

Synopsis

vsenable

Description

When the **vsenable** command is run, UNIX system files are modified to allow the voice system to be automatically started when the CONVERSANT VIS is rebooted. By default, the voice system is installed with the automatic startup enabled. If there were any non-fatal problems during installation, the voice system is still installed but it has not enabled for automatic startup at system reboot. After the installation problems have been cleared, use **vsenable** to enable automatic voice system startup at reboot.

Example

vsenable

See Also

"vsdisable"

vusage

Name

This command displays the current load on the voice system.

Synopsis

vusage

Description

The **vusage** command enables the voice system administrator to determine the load on the voice system. It queries the voice system and prints the response on the screen, indicating the maximum number of channels in the system and the number of channels playing or coding, and the maximum number of buffers and the number in use.

Example

The following is an example of the **vusage** command and sample output.

```
$ vusage

    Max (Current) Speech Buffers used:   0   (0)
    Max (Current) Chans playing/coding:  0   (0)

$
```

Warnings

The voice system must be running to execute this command.

See Also

"display channel"
"sysmon"

xferdip_off

Name

This command deactivates the bridging capability.

Synopsis

xferdip_off

Description

The **xferdip_off** command deactivates the bridging capability. If the xferdip is running, this command stops it.

Example

xferdip_off

xferdip_on

Name

This command activates the bridging capability.

Synopsis

xferdip_on

Description

The **xferdip_on** command activates the bridging capability. If the VIS is running, this command starts the xferdip.

Example

xferdip_on

Index